Lu Dai

DNS Query Prediction

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master’s Thesis

16 March 2024

PREFACE

This thesis focuses more on the engineering approach to benefit network applications by predicting the DNS query event, rather than an academic study.

The idea to predict DNS query event was triggered several years ago when I noticed a new HTML syntax named dns-prefetch was introduced. I thought it was network efficiency-wise after a quick experiment. But the limitations were obvious if I can say so unfairly. First, it's for web browsers dedicatedly, of course, other network applications can borrow this design also. Secondly, it requires developers to specify the domain names in the source code.

I was thinking that: is it possible to automatically learn which host the application will connect to, purely from network traffic without the involvement of the user application. It provides a better possibility to do something like "dns-prequery", which can benefit all types of network applications without their awareness. This question means different things in different time scales. In days and weeks, it's about user profiling. But I was more focused on smaller time scales like seconds or milliseconds, that's where we can accelerate DNS service. After reading several network traffic streams, my intuition told me it seems possible, at least I can easily recover the user's browsing history in my brain based on the packets, which means a pattern exists. But the traffic is extremely noisy, a situation where the human brain can easily defeat the computer. So, my conclusion was: possible but challenging.

At that moment, I was working as an embedded software engineer to develop a network router, it was a pity that I didn't have the chance to explore even a PoC on that product. So, I chose it as my thesis topic to satisfy my curiosity. While doing the thesis, I encountered a lot of difficulties in learning knowledge from the extremely noisy raw DNS traffic data, which is a one-dimensional observation to a high-dimension model. But anyway, it's more focused on a feasible engineering solution, rather than academic research.

Espoo, March 2024
Lu Dai

Abstract

Author:	Lu Dai
Title:	DNS Query Prediction
Number of Pages:	34 pages
Date:	16 March 2024
Degree:	Master of Engineering
Degree Programme:	Information Technology
Professional Major:	Networking and Services
Supervisors:	Peter Hjort, Lecturer

DNS query event as the prerequisite of network connection, introduces extra time latency to complex network application which involves a group of network activities. The latency cannot be optimized in the pipeline of network interactions in current DNS framework. This thesis proposed a prediction solution and corresponding learning method to reduce the overall latency. By estimating conditional probability, which is a widely used metric in natural language processing to solve the “word association” problem, which is similar to this problem, this thesis proposed a learning method to learn query associations from DNS traffic. This thesis also proposed a standalone prediction solution and an integrated prediction solution to cooperate with current DNS cache mechanism, to accelerate DNS service by predicting.

Keywords: DNS, discrete time series, statistics

Contents

List of Abbreviations

DNS	Domain Name System
IP	Internet Protocol
CDN	Content Delivery Network
HTML	Hyper-Text Markup Language
CSS	Cascading Style Sheets
NIC	Network Interface Card
ISP	Internet Service Provider

1 Introduction

The Domain Name System (DNS), as a fundamental component of the internet infrastructure, alleviates the need for users to memorize lengthy Internet Protocol (IP) addresses. It achieves this by mapping human-readable host domain names to computer-readable IP addresses, playing a crucial role as internet applications grow increasingly complex. Since its inception by Mockapetris [1], DNS has undergone various enhancements, such as DNSSec [2] for improved security, DNS-over-TCP [3] for enhanced transaction efficiency, DNS-over-HTTPS [4] to leverage web browser capabilities, and DNS-over-TLS [5] for the utilization of lightweight cryptographic infrastructure.

Despite the cache mechanism outlined in [1], the deployment of DNS services on the internet exhibits diverse configurations. Contemporary network applications extensively employ Content Delivery Networks (CDNs) [6] and load balancing [7], dynamically offering host domain name resolution services based on geographic location and server cluster load. In all forms of DNS and its variants, the fundamental interaction involves a query and a response, where the query seeks the IP address of a host, and the response provides that information. Notably, each query is handled independently, without consideration for prior queries or knowledge of correlations between hosts within the DNS schema.

Empirically, correlations between hosts are pervasive on the internet. It requires no specialized knowledge for an ordinary person to suspect a connection between *www.youtube.com* and *accounts.youtube.com*. Research [11] demonstrates that hyperlinks in hypertext applications, including web applications, unveil relationships between documents. As these documents exhibit correlation, the hosts providing them also share correlation from the resource-accessing perspective. Section 2 elaborates on why and how these correlations exist, using web applications as an example, but this pattern applies to any type of network application. Armed with the knowledge that host domain names, such as *accounts.youtube.com* and *www.youtube.com*, are correlated, the question arises: can we expedite the loading of the YouTube homepage? By

predicting the query for *accounts.youtube.com* when handling the query for *www.youtueb.com,* after we see that *accounts.youtube.com* always come after *www.youtueb.com* in the traffic. Figure 1 shows the concept of learning and usage of the prediction.

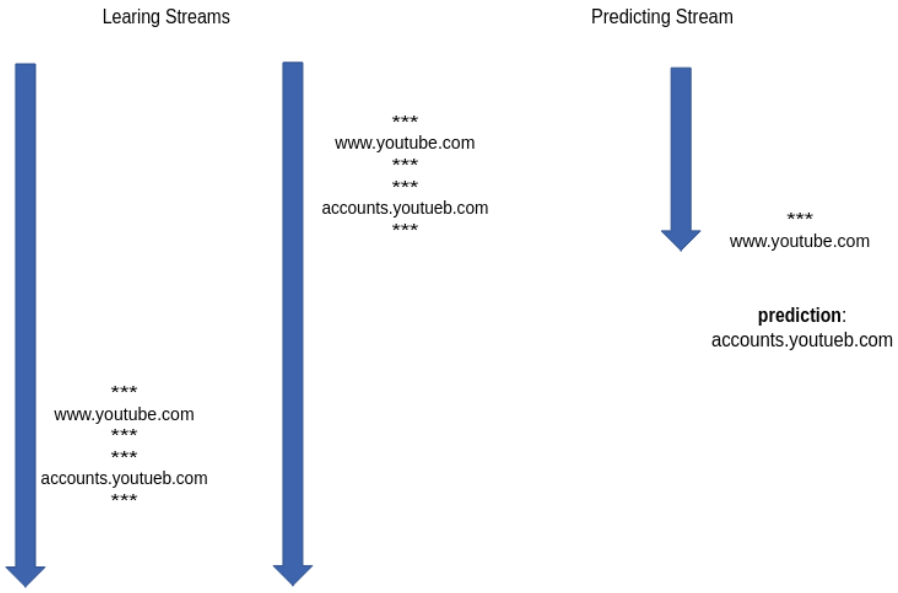

Figure 1: Learning and Prediction Usage

In all DNS use cases, the time cost of a DNS transaction to resolve a host domain name to its corresponding IP address is considered an overhead in accessing a host. Whenever a user connects to a host with a given domain name, DNS is invoked as a prerequisite, incurring a typical cost of three to five milliseconds. The overhead accumulates when an internet application requires resources from multiple hosts, resulting in increased DNS time costs for more host connections. This accumulation becomes inevitable as modern internet applications distribute resources and logic processes across multiple servers. For instance, a web service may store its Hyper-Text Markup Language (HTML) document, JavaScript document, images, and authentication processes on different hosts. Compounding the issue is the sequential resolution of these prerequisites, where

a user must resolve the host to obtain the HTML document before parsing it and determining where to retrieve the referenced JavaScript document. As internet applications grow in complexity and necessitate more hosts, the DNS time cost becomes a prominent challenge.

Efforts have been made to reduce the time cost of DNS. [1] proposed a cache mechanism to avoid redundant queries for the same host, while [8] introduced a pre-fetch mechanism in web browsers enabling users to query in advance. This research explores a novel approach by treating a hostname as a member within an association, rather than a standalone variable. Assuming hosts, such as *www.youtube.com* and *accounts.youtube.com*, belong to the same association, the DNS query overhead for the latter can be reduced or eliminated by predicting it based on the occurrence of the former. This research introduces a probabilistic approach to decrease overall time costs by predicting the next DNS query event on a micro time scale, leveraging knowledge of correlations derived from DNS raw traffic through conditional probability estimation.

1.1 Related Works

The problem this thesis aims to solve is like the word association [16] [17] in natural language processing (NLP) field. “In natural language, words are not combined randomly into phrases and sentences, constrained only by the rules of syntax.” [16]. Compare DNS traffic stream to text document, and DNS query to word, cooccurring of queries to cooccurring of words, techniques used in NLP can also be used to solve this problem. Some adaptations are required to meet the DNS traffic stream as a discrete time series. The core metric to evaluate the association of two “words” used in the thesis is conditional probability, refers to [16] section “Conditioning on fixed marginal frequencies”. In [16], author used one word as a condition, estimate the probability of another word, as variable, occurred in the same row. In this thesis, we used one query as a condition, to estimate another query occurred in a narrow temporal window.

1.2 Structure

Section 2 outlines common features in web applications affecting DNS query behaviour, a pattern applicable to other network applications, and constructs a probabilistic model for DNS query sequences, accompanied by relevant experimental results. Section 3 details the methodology for preprocessing and processing DNS raw traffic to learn and feature host association knowledge. Section 4 presents the system design for predicting DNS queries, to put the knowledge into use to reduce the overall time cost of DNS service.

All data and source code are available on the website *www.github.com/exlud/pcap-dns-statistic*.

2 Query Pattern and Model

This chapter delineates the DNS query behavioural pattern from the perspective of user applications. Despite the autonomous handling of DNS queries, there exists a cohesive and consistent grouping of host domains associated with each user application. This signifies that these queries exhibit a spatial-temporal adjacency relationship. Given foreknowledge of the user application, it becomes possible to anticipate a specific group of queries within a narrow temporal window in the DNS traffic stream. The spatial-temporal adjacency pattern thus presents an opportunity to cluster host domains and predict DNS query events.

In this section, a probabilistic model is formulated to depict the interdependence of queries. This is achieved through a form of conditional probability, where one query serves as the dependent variable, and another as the independent variable. The estimation of conditional probability, serving as a parameter of the model, is derived from the learning process applied to raw DNS traffic data. This estimation, in turn, functions as a metric of distance for clustering. The model incorporates certain empirically based assumptions to generalize across diverse user applications and to streamline the complexity inherent in the model.

Subsequent sections of this thesis employ the term "query" as a shorthand reference to "DNS query," "observation" for "DNS query event observation," and "host" in reference to "host domain name."

2.1 Query and Observation

A query event is typically initiated by the user application to resolve a given host to the corresponding IP address before establishing a transport layer connection to access the required resource, which may be a document or a service. The associated observation manifests as a DNS message within the network traffic, querying for that specific host.

Consider the example of the *www.youtube.com* webpage. To retrieve the annotated HTML document resource (R), a prerequisite query event (Q) is necessary before initiating a Transmission Control Protocol (TCP) connection to the web server. The term "resource" encompasses not only HTML documents but also services such as database routines, authentication routines, images, ad tracking service, etc. R sufficiently causes Q as the effect from the perspective of causality as shown in formula (1).

$$R(www.youtube.com/index.html) \rightarrow Q(www.youtube.com) \quad (1)$$

Asymmetrical mapping between R and Q is prevalent, where R uniquely maps to one Q, but a single Q may map to multiple R. An illustration of this can be found in formula (2) and (3) when various resources necessitate the same initial query.

$$R(www.youtube.com/player.js) \rightarrow Q(www.youtube.com) \quad (2)$$

$$R(www.youtube.com/authentication) \rightarrow Q(www.youtube.com) \quad (3)$$

Complicating matters, especially in web applications, is the fact that static resources, like JavaScript documents, can be locally cached in web browsers. Consequently, R may not always cause the corresponding Q. This causality relationship with probability is annotated as formula (4).

$$R(host/resource) \overset{prob.}{\rightarrow} Q(host) \quad (4)$$

In numerous instances, the occurrence of the Q event results in an observation (q) in the form of a DNS message within network traffic. However, this is not always the case due to the DNS record cache mechanism. The processing of the query event occurs sequentially, involving the user application, operating system (OS), stub resolver, recursive resolver, and root resolver. Nearly all endpoints in this chain have the capability to cache DNS records. Depending on the observer's location, the DNS message as an observation may not always be present. For example, when observing on the Network Interface Card (NIC) of a computer and the OS has a valid and cached IP address for a specific host name, the DNS

message becomes unnecessary unless insisted upon by the user application. The causality relationship between R, Q, and q is annotated as formula (5).

$$R(host/resource) \xrightarrow{prob.} Q(host) \xrightarrow{prob.} q(host) \quad (5)$$

2.2 Queries and Sequence

A network service may utilize a group of hosts to distribute subroutines, implying that a network activity may invoke multiple queries within a narrow temporal window. The sequence of DNS queries, serving as a profile of the network activity, carries contextual significance.

In modern network applications, it is common to organize a cluster of hosts to serve a single user service. For instance, a website might utilize a group of hosts to deliver various types of resources such as HTML, JavaScript, CSS, images, and videos required for a single web page. Similarly, an IoT application might distribute different services like authentication, diagnostics, command & control, and firmware upgrades across a group of hosts.

Given a user activity (C) and the corresponding group set (Sc), the posterior probability of those hosts in the group should sum to one, and the conditional probability for any two of them should also sum to one, as annotated in formulas (6) and (7).

$$P(h_i \mid C) = 1, h_i \in S_c \quad (6)$$

$$P(h_i \mid C, h_j) = 1, h_i, h_j \in S_c \quad (7)$$

Empirically, every network service typically possesses at least one distinctive element within its corresponding host group. This emphasizes the constraint of the spatial-temporal space concerning where and when the observer collects the data. By utilizing this unique element as a condition, other elements within the

same group can be identified through significant conditional probability estimation, as illustrated in formula (8).

$$P(h_i \mid h_j)=1, h_i, h_j \in S_{cm}, h_j \notin S_{cn}, m \neq n \quad (8)$$

The DNS query sequence, functioning as the temporal dimension observation, inherits the aforementioned characteristic, revealing repetitive patterns where two events occur within a narrow temporal window in a single DNS traffic stream, as depicted in formula (9), where tau (τ) serves as a hyperparameter defining the window size. Generally, a larger window size tends to increase the noise level, while a smaller window size enhances the occurrence of positive false instances in the learning process. Further detailed discussion on this matter is provided in section 3.3.

$$P(h^i_{T\pm\tau} \mid h^j_T)=1 \quad (9)$$

Two types of annotations are utilized in this thesis to represent a discrete time sequence, as illustrated in Figure (2). The latter annotation method provides a more vivid depiction of the time intervals.

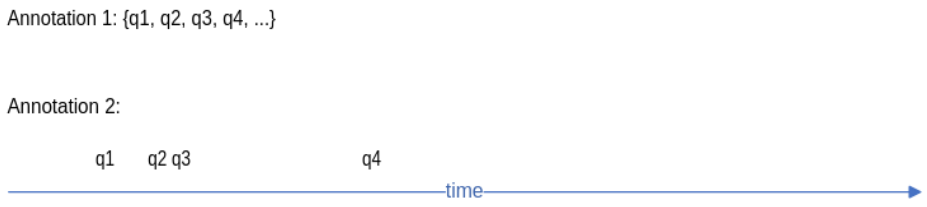

Figure 2: Annotations of Sequence

This thesis operates under the assumption that user activities are independent of each other. However, it is important to note that this assumption is not entirely accurate, given the complex nature of higher-level user behaviours. Nonetheless, making this assumption aids in simplifying the probability description when multiple user activities intertwine within a single sequence. Without prior knowledge of whether an element is unique or not, the conditional probability is described as Formula (10).

$$P(h_i \mid h)=\frac{\sum P(C_m)}{\sum P(C_n)}, h \in C_n, \{h, h_i\} \in C_m (10)$$

Certainly, when the condition uniquely corresponds to only one user activity, it becomes feasible to cluster hosts into groups through posterior probability estimation.

2.3 Query Dependency

In web applications, documents utilize references, links, or citations to establish connections with other documents, and this pattern extends to other types of network applications as well. When a user application interacts with a host, the information provided by the host can be regarded as a resource, which is a general form of a document. Resources can vary widely, including HTML/CSS/JS documents, images, videos, database services, authentication services, ad-tracking services, and so on. These resources often exhibit dependency relationships, where one resource requires another, as demonstrated in formulas (11) and (12).

$$R(www.youtube.com/html) \rightarrow R(fonts.googleapis.com/css) (11)$$

$$R(www.youtube.com/authentication) \rightarrow R(accounts.youtube.com/service) (12)$$

By cascading the dependency relationships, a resource dependency tree (RDT), as illustrated in Figure (3), can be utilized to describe the user activity (C).

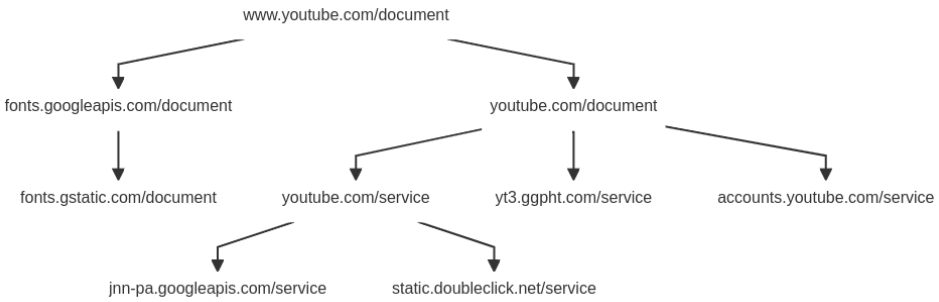

Figure 3: RDT (www.youtube.com)

Taking into account the fact that accessing a resource necessitates a DNS query event, as discussed in section 2.1, when given the user activity where the resource can uniquely map to one query event, a query dependency tree (QDT) can be constructed, as illustrated in Figure (4).

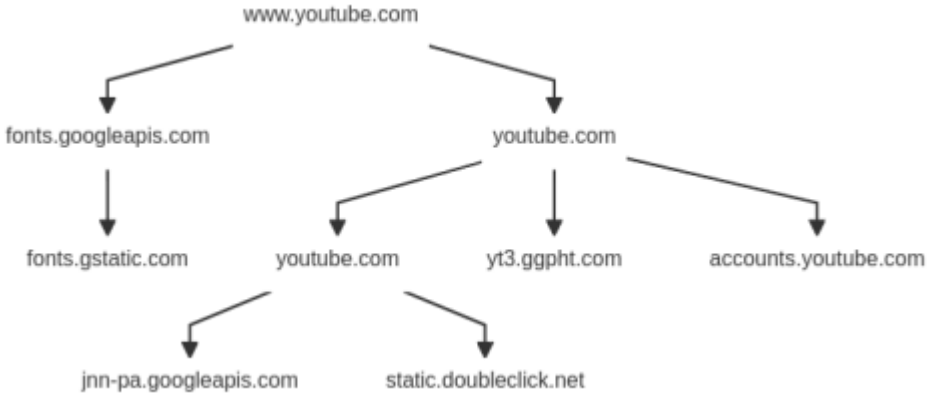

Figure 4: QDT (www.youtueb.com)

In the above example, suppose the given user activity is to load the home page of the website www.youtube.com. The root node Q(www.youtube.com) depends on a group of other queries. This dependency relationship can be represented through conditional probability, where these queries are adjacent to each other in a micro timescale, and the dependent query must occur prior in time to the dependency.

Generally speaking, queries contained within one QDT are more likely to be observed as neighbours in a narrow temporal window. However, in reality, the

sequence can be influenced by several other mechanisms, including cache effects, query multiplexing, and query variants.

2.4 Cache Effect

Apart from the widely implemented document caching mechanism in web browsers, which caches documents and eliminates DNS queries, the DNS cache mechanism affects query behaviour in a broader scope for all network applications.

Storing frequently used JavaScript or CSS documents locally in the browser is a reasonable and efficient choice. In such cases, the process to load the home page of www.youtube.com may not trigger the subroutine to access fonts.googleapis.com because the document is already available locally. This implies that in one experiment, part of the QDT, which is determined by the RDT, may not be observable.

Most operating systems and DNS stub resolvers may enable the feature to cache DNS records, leading to a possibility where a query (Q) may not trigger the corresponding observation (q), as the query event might be intercepted by the operating system or stub resolver in the middle.

The unified representation of cache effects introduces an additional parameter to the probability of query events, as shown in formulas (13) and (14). In this manner, the RDT and QDT are treated as persistent and repetitive, with observations (q) being influenced by a random sequence in the meantime. The value of the random sequence can be either one or zero, as defined in formulas (15) and (16).

$$R(host/resource) \rightarrow Q(host) \overset{\text{prob.}}{\rightarrow} q(host) \quad (13)$$

$$P(q(A)) = \alpha \cdot P(Q(A)) \quad (14)$$

$$P(\alpha=1)=p(15)$$

$$P(\alpha=0)=1-p(16)$$

The free parameter p represents a pseudo-sequence where its value is close to one at the first occurrence of a user activity, and close to zero afterward. The cache effect introduces a scalar to the probability estimation but does not alter its characteristics. In other words, it does not change the correlation between hosts, as demonstrated in formula (17).

$$P(h_i \mid h_j)=\frac{P(h_i,h_j)}{P(h_j)}=\frac{\alpha^2 \cdot P(Q_{h_i}) \cdot P(Q_{h_j})}{\alpha \cdot P(Q_{h_j})} (17)$$

.

2.5 Query Multiplex

A host can serve multiple services and can be accessed in various ways. For instance, a host storing images for one website might also store videos for another website simultaneously. Similarly, a host providing HTTP service can also serve as an ICMP destination at the same time. This multiplexing of hosts makes the dependency relationship volatile, particularly distinguished in different user activity contexts.

Examples of host multiplexing are abundant in the real world. Consider three different scenarios: 1) marking www.youtube.com as a favourite in the web browser and then opening the web browser; 2) pinging www.youtube.com in the command line to test network connectivity; 3) visiting the webpage of www.youtube.com. Only in the third scenario does the dependency relationship exist.

To mitigate the effect of multiplexing on observations, one feasible approach is to cluster sequences based on their features in a medium timescale, as demonstrated in previous research [14]. In this thesis, sequence clustering was

performed manually. Through this method, training data are selected to have the same user activity, as illustrated in Figure (5).

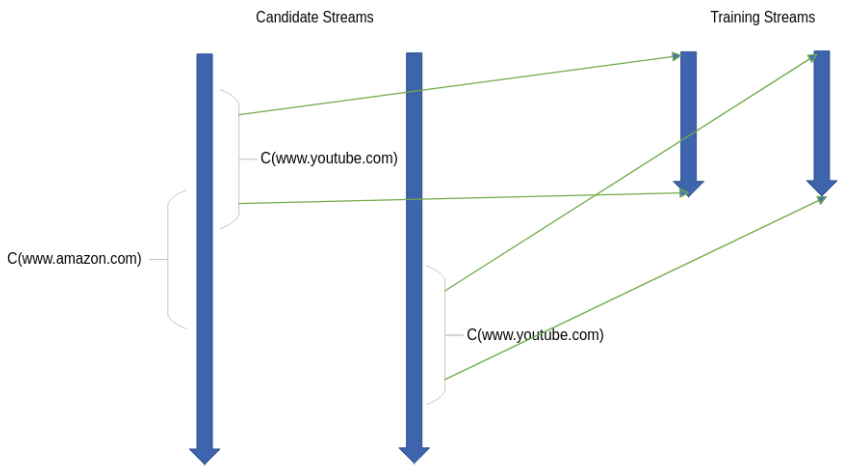

Figure 5: Mitigate Multiplexing

2.6 Query Variants

The widely used CDNs and load balancing mechanisms introduce variations in queries at runtime. It is common for a network service to have backend logic, as illustrated in List (1) and (2).

```
def dynamic_cdn_server
  if client from Asia:
    host = img-asia.foo.bar
  else if client from Europe:
    host = img-eu.foo.bar
  else:
    host = img.foo.bar
  return host
end
```

List 1: Example of CDN Backend Logic

```
def dynamic_load_balance
```

```
  host = round-robin(db1.foo.bar, db2.foo.bar, db3.foo.bar)
  return host
end
```

List 2: Example of Load Balancer Backend Logic

The variation caused by CDNs is relatively easier to address in practice, especially if the learning materials are generated within the same Autonomous System (AS) or geographic location, which is often achievable. In this thesis, DNS traffic was collected in a single geographic location, which helps mitigate CDN-induced variations.

On the other hand, load balancing introduces greater complexity to the dependency relationship. Resources can be dynamically fetched from any one of a group of hosts, as depicted in formula (18), leading to potential changes in observations over time. Unfortunately, this thesis cannot fully resolve this issue, and the variants may be treated as noise in the learning process. However, given the prediction usage targeted by the thesis, this limitation is acceptable due to the inherently unpredictable nature of load balancing mechanisms.

$$R(A) \to Q(A) \overset{\text{prob.}}{\to} q(A), A \in \{A_1, A_2, A_3 \ldots\} \quad (18)$$

Tree, Graph and Association

Given a Query Dependency Tree (QDT), an association can be derived to depict the connection of hosts. For example, consider the association A(www.youtube.com) obtained from the tree T(www.youtube.com), which includes elements such as www.youtube.com, font.googleapis.com, font.gstatic.com, youtube.com, accounts.youtube.com, yt3.ggpht.com, jnn-pa.googleapis.com, and static.doubleclick.net.

This association can be leveraged to predict DNS query events. When receiving a query q(www.youtube.com), confidence is increased to expect other elements

in A(www.youtube.com), such as q(accounts.youtube.com), depending on the application.

To learn the association from DNS traffic, this thesis utilizes an undirected graph method weighted by probability. The vertices (V) of the graph are defined as hosts, and the edges (E) are determined by the conditional probability of two vertices being connected, which is higher than a given threshold, as described in formulas (19) and (20).

$$V \subseteq \{host\} \quad (19)$$

$$E \subseteq \{\{h_1, h_2\} \vee h_1, h_2 \in V, h_1 \neq h_2, P(h_1 \mid h_2) > \epsilon \; OR \; P(h_2 \mid h_1) > \epsilon\} \quad (20)$$

Given an example of conditional probability estimation in Table (1), the learned vertices and edges can be derived using formulas (21) and (22). This results in the learned vertices and edges that form the undirected graph representing the association among hosts based on the conditional probabilities observed in the DNS traffic.

Var. / Con.	h_1	h_2	h_3
h_1	N/A	0.9	0.1
h_2	0.6	N/A	0.3
h_3	0.2	0.2	N/A

Table 1: Example of Conditional Probability Estimation

$$V \subseteq \{h_1, h_2, h_3\} \quad (21)$$

$$E \subseteq \{\{h_1, h_2\}\}, \epsilon = 0.5 \quad (22)$$

Using the described method with a conditional probability threshold and distance, an association for a given host can be obtained from DNS traffic by estimating the probability. This association consists of hosts that can be connected to within the specified number of hops.

For example, consider a conditional probability threshold of 0.5 and a maximum distance of 2 hops from the given host h_1. We can iterate through the graph and identify all hosts that can be connected to h_1 with a probability greater than the threshold within given hops. The resulting set of hosts forms the association for h_1.

Using this approach, the association for each host can be determined based on the observed conditional probabilities and the specified parameters of threshold and hops. This enables the learning process in section 3.

3 Learning Process

This section describes the learning process, which includes data pre-processing, conditional probability estimation, and propagation, all aimed at learning host associations. Experiments results are alongside.

3.1 Pre-Process

Raw data consists of DNS traffic captured using tcpdump on a laptop within the same autonomous system (AS) over several days, stored in Wireshark pcap file format.

One of the challenges in the learning process is memory efficiency. With potentially billions of DNS records, building a conditional probability structure at this scale would be impractical. To address this issue, pre-processing involves aggregating traffic flows that share the same user activity. While unsupervised clustering methods, such as those used in [10], can solve this problem by learning DNS sequence features in a medium time scale, this thesis chooses to manually select training data to focus on a limited number of user activities. Both training and test data are manually labelled for performance evaluation, although not for supervised learning. Data and labels are available alongside the source code.

3.2 Estimate Probability

This step aims to discover the correlation between two hosts. Estimating the probability for one variable with multiple conditions requires a larger training data volume and higher sample rate, which can be less robust in practice when the sample is sparse. This thesis uses only one condition, along with a method described in section 3.4 to find connected components in a graph to compensate for potential side effects.

Given a host as a condition, the conditional probability estimation for a host variable is defined in formula (23), where |H| is the space for all DNS records in the learning material, N(hi) is the occurrence of hi in the sequence, and N(hi, hj) is the joint occurrence of hi and hj within a given time distance, which is a hyperparameter.

To enhance robustness when the sample volume is small or when a host has a small occurrence count, Laplace smoothing is applied to prevent unfair advantages for host pairs that occur together but with a small count, as defined in formula (24).

$$P(H_i \mid H_j) = \frac{N(h_i, h_j)}{N(h_j)}, H \in H \quad (23)$$

$$P_{smooth}(H_i \mid H_j) = \frac{N(h_i, h_j) + \alpha}{N(h_j) + \alpha d}, H \in H \quad (24)$$

Indeed, the additive used in Laplace smoothing imposes a higher penalty on hosts with fewer occurrences, thereby mitigating the influence of sparse data. However, as the occurrence of hosts increases, the effect of the additive diminishes.

Host pairs (h_i, h_j) are considered connected when their conditional probability exceeds a certain threshold (t). Thus, given a host (h) and a threshold (t), a group of hosts can be learned from the DNS traffic by satisfying the condition outlined in formula (25), suggesting potential connections among these hosts. This threshold-based approach allows for the identification of host associations based on their observed conditional probabilities, facilitating the extraction of meaningful relationships from DNS traffic data.

$$learned(h, t) = \{\{h_j\} \mid max(P(h_j \mid h), P(h \mid h_j)) > t\} \quad (25)$$

Table (2) shows the experiments result on a given user activity (loading www.youtube.com web page) with different threshold.

learned(www.youtueb.com, 0.8)	
target	learned
i.ytimg.com	
fonts.googleapis.com	
fonts.gstatic.com	
googleads.g.doubleclick.net	
static.doubleclick.net	
jnn-pa.googleapis.com	jnn-pa.googleapis.com
accounts.youtube.com	
www.gstatic.com	
www.google.fi	
play.google.com	
yt3.ggpht.com	
load balanced	
rr2---sn-q4flrnes.googlevideo.com	
lh5.googleusercontent.com	
rr1---sn-qo5-ixas.googlevideo.com	
	rr4---sn-5hne6nzd.googlevideo.com
	rr2---sn-qo5-ixas.googlevideo.com

learned(www.youtueb.com, 0.7)	
target	learned
i.ytimg.com	i.ytimg.com
fonts.googleapis.com	
fonts.gstatic.com	
googleads.g.doubleclick.net	
static.doubleclick.net	
jnn-pa.googleapis.com	jnn-pa.googleapis.com
accounts.youtube.com	
www.gstatic.com	www.gstatic.com
www.google.fi	
play.google.com	
yt3.ggpht.com	yt3.ggpht.com
load balanced	
rr2---sn-q4flrnes.googlevideo.com	
lh5.googleusercontent.com	
rr1---sn-qo5-ixas.googlevideo.com	
	rr4---sn-5hne6nzd.googlevideo.com

	rr2---sn-qo5-ixas.googlevideo.com

learned(www.youtueb.com, 0.6)	
target	learned
i.ytimg.com	i.ytimg.com
fonts.googleapis.com	
fonts.gstatic.com	
googleads.g.doubleclick.net	
static.doubleclick.net	
jnn-pa.googleapis.com	jnn-pa.googleapis.com
accounts.youtube.com	
www.gstatic.com	www.gstatic.com
www.google.fi	
play.google.com	
yt3.ggpht.com	yt3.ggpht.com
load balanced	
rr2---sn-q4flrnes.googlevideo.com	
lh5.googleusercontent.com	
rr1---sn-qo5-ixas.googlevideo.com	
	rr4---sn-5hne6nzd.googlevideo.com
	rr2---sn-qo5-ixas.googlevideo.com

learned(www.youtueb.com, 0.5)	
target	learned
i.ytimg.com	i.ytimg.com
fonts.googleapis.com	
fonts.gstatic.com	
googleads.g.doubleclick.net	
static.doubleclick.net	
jnn-pa.googleapis.com	jnn-pa.googleapis.com
accounts.youtube.com	
www.gstatic.com	www.gstatic.com
www.google.fi	
play.google.com	
yt3.ggpht.com	yt3.ggpht.com
	ocsp.pki.goog
	r3.o.lencr.org
load balanced	
rr2---sn-q4flrnes.googlevideo.com	
lh5.googleusercontent.com	
rr1---sn-qo5-ixas.googlevideo.com	

	rr4---sn-5hne6nzd.googlevideo.com
	rr2---sn-qo5-ixas.googlevideo.com

learned(www.youtueb.com, 0.4)	
target	learned
i.ytimg.com	i.ytimg.com
fonts.googleapis.com	
fonts.gstatic.com	
googleads.g.doubleclick.net	googleads.g.doubleclick.net
static.doubleclick.net	
jnn-pa.googleapis.com	jnn-pa.googleapis.com
accounts.youtube.com	
www.gstatic.com	www.gstatic.com
www.google.fi	
play.google.com	
yt3.ggpht.com	yt3.ggpht.com
	ocsp.pki.goog
	r3.o.lencr.org
	sync-1-us-west1-g.sync.services.mozilla.com
	detectportal.firefox.com
load balanced	
rr2---sn-q4flrnes.googlevideo.com	
lh5.googleusercontent.com	
rr1---sn-qo5-ixas.googlevideo.com	
	rr4---sn-5hne6nzd.googlevideo.com
	rr2---sn-qo5-ixas.googlevideo.com

Table 2: Probability Connection

The precision rate and recall rate, as defined in equations (26) and (27) respectively, are commonly used metrics to evaluate the performance of learning processes based on probability estimation. Precision rate quantifies the accuracy of positive predictions made by the learning process, recall rate quantifies the completeness of positive predictions made by the learning process.

$$precision = \frac{|target \cap learned|}{|target|} (26)$$

$$recall = \frac{|target \cap learned|}{|learned|} (27)$$

Figure (6) illustrates the performance of learning the user activity C(www.youtube.com) at different threshold parameters.

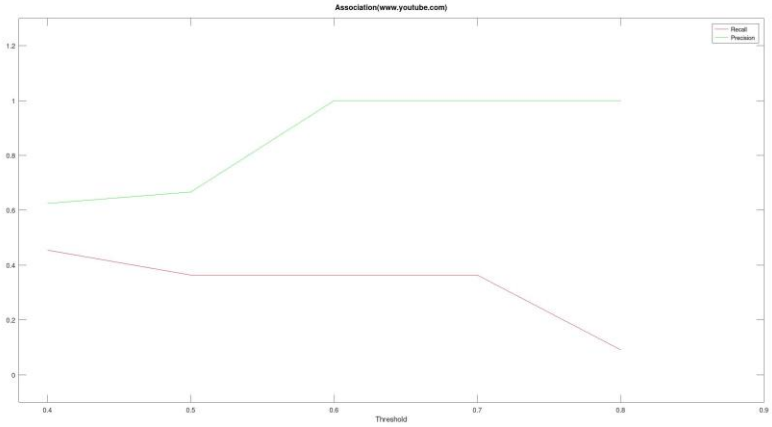

Figure 6: Precision and Recall Depends on Threshold

The result shows that higher threshold increases recall rate but decreases precision rate in the meantime. Recall rate did not increase after the threshold goes down to 0.7.

The results indicate a high precision rate but a low recall rate. As the threshold decreases, allowing for more connections to be considered, the recall rate improves but at the cost of an increased false positive rate. Notably, when the threshold is lowered to 0.6, the false positive rate rises significantly. This suggests that while lowering the threshold improves recall, it also leads to a higher likelihood of falsely identifying connections among hosts that may not actually be related to the user activity. Therefore, a trade-off between precision and recall needs to be carefully considered when selecting the threshold for host association learning.

3.3 Propagate Connection

Propagate the connection is very similar to search for a sub-connected component in the graph, starting from one vertex, where all other vertices in the component are reachable within a given number of hops (edges). For example, if {a, b} are connected pair, and {b, c} are also connected pair, then {a, c} can be treated as connected within two hops, even if {a, c} do not meet the requirement of probability significance directly.

One reason for performing this operation is to mitigate the side effects of the hyperparameter time window. The time window for one user activity varies depending on network quality, server performance, and application complexity. A larger time window introduces more noise, increasing false positives, while a smaller time window increases precision but reduces recall. Since precision rate is preferred over recall rate, the time window is relatively small in section 3.2. To compensate for this, this section aims to increase the recall rate by propagating connections.

Another reason, as discussed in sections 2.1 and 2.5, is that some hosts may uniquely belong to one user activity or belong to a limited set of user activities, resulting in more significant conditional probabilities with other hosts in the same user activity. Propagating connections allows for the utilization of this characteristic.

Experiments comparing methods in section 3.3 and section 3.2 suggest that propagation helps to increase the precision rate at the cost of decreased recall rate. This trade-off should be carefully considered when selecting the appropriate method for learning host associations based on conditional probability estimation.

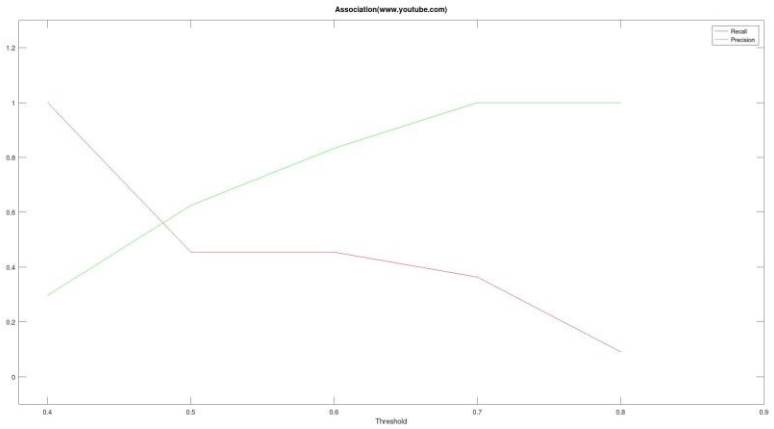

Figure 7: Precision and Recall after Propagation

The reason why the recall rate goes up to 1 when the threshold is set as 0.4 is due to the additive used in Laplace smoothing, which is set to 3. This additive increases the probability for host pairs that occur at least 2 times to be treated as significantly connected. Consequently, a lower threshold allows for more host pairs to be considered as connected, resulting in a higher recall rate.

As the recall rate and precision rate tend to go opposite directions, it is reasonable to assume that when the training data volume increases, better performance can be achieved. With more data, the model can better capture the underlying patterns and associations among hosts, leading to improved performance in terms of both precision and recall.

Experiments also show that this method helps mitigate the time effects of the hyperparameter of the time window size. Connecting hosts by hops is more efficient compared to increasing the window size, as illustrated in Figure (8). This suggests that propagating connections by hops is an effective strategy for capturing associations between hosts over varying time windows, without significantly increasing computational complexity or noise in the data.

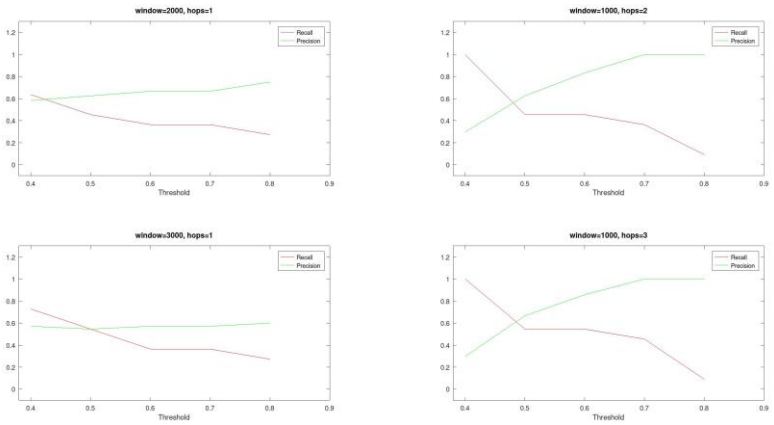

Figure 8: Comparison of multi hops and bigger window

4 Prediction Application

To put the knowledge learned in section 3 into use, the section proposed an application design which coexists with cache mechanisms, operating within the current DNS service framework without requiring awareness of user applications. Importantly, it offers benefits to all types of network applications, which is a significant advantage compared to dns-prefetch, which primarily benefits web applications only.

Deployments

According to the DNS framework design in [1], the hierarchical system can be illustrated as Figure (7).

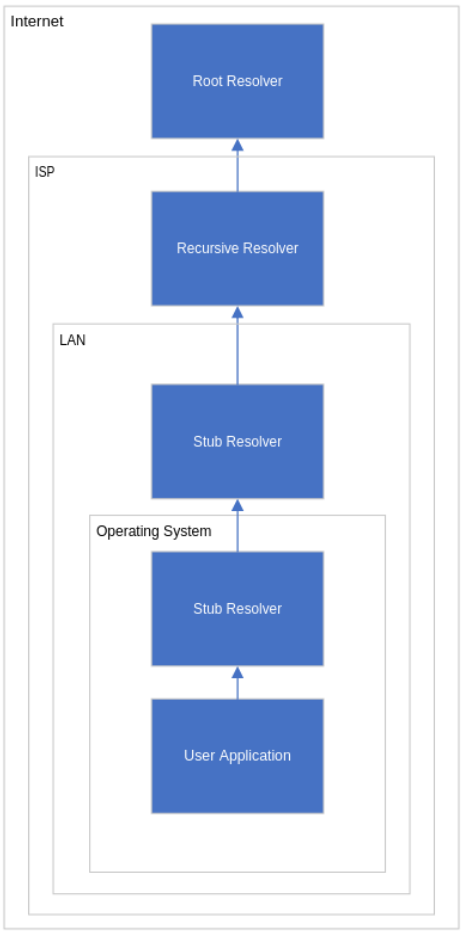

Figure 7: DNS Hierarchical System

In this system, stub resolvers are typically deployed in proximity to the user application, such as within the operating system's network protocol stack, on the home router, or within the accessing layer device of the internet service provider (ISP). These stub resolvers commonly facilitate the DNS cache mechanism, as depicted in Figure (8).

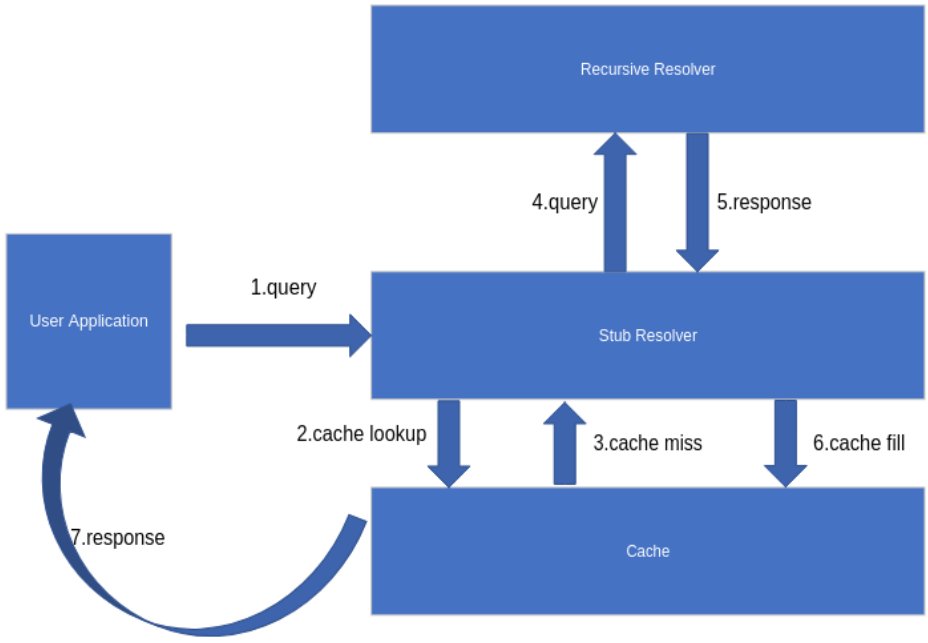

Figure 8: DNS Cache Mechanism in Stub Resolver

Sequence diagram of these three participants illustrated as Figure (9).

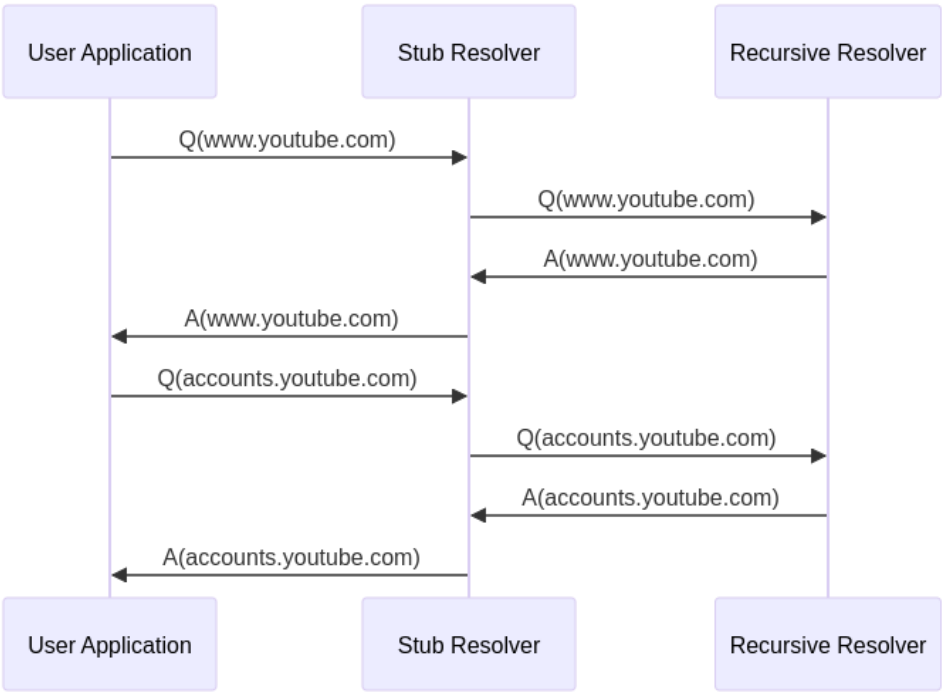

Figure 9: Sequence Diagram of DNS interactions

The prediction application can stand alone or integrate into cache mechanism, illustrated in Figure (10) and Figure (12).

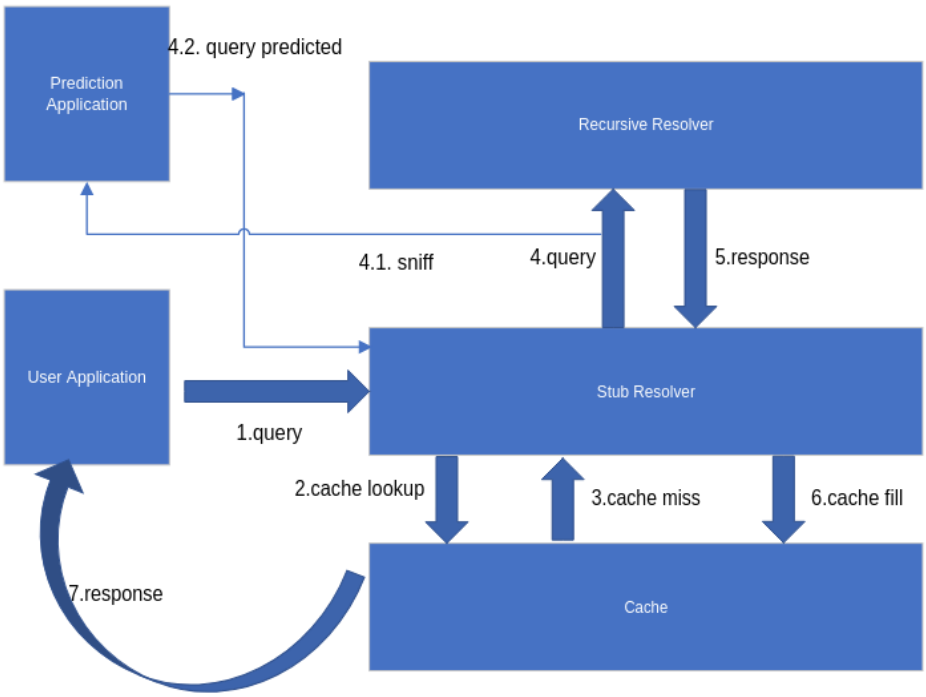

Figure 10: Standalone Prediction Application

The standalone prediction system sniffs DNS query messages on the NIC interface, a task feasible by enabling promiscuous mode, and predicts DNS records in parallel. This non-intrusive design does not necessitate any modifications to the user application, stub resolver, or DNS infrastructure. The sequence diagram depicting the interactions among these four participants is illustrated as Figure (11). In comparison to Figure (9), the interactions between the stub resolver and recursive resolver are altered, with the predicted query occurring earlier than previously. This modification results in a benefit to user applications, namely, a shorter response time, as the DNS record is readily available in the local cache.

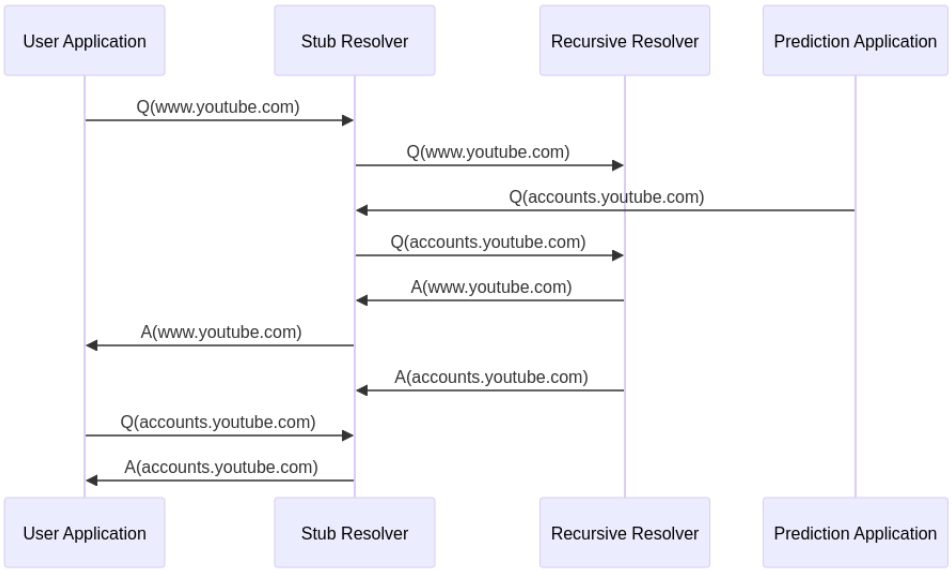

Figure 11: Sequence Diagram of Standalone Prediction

Figure (12) illustrates an alternative approach to integrate prediction into the existing cache management process. In the event of a cache miss for one host, aside from filling up that particular host, other predicted hosts also need to be filled. Unlike the standalone method, this design necessitates changes in DNS

stub resolver applications. However, the benefit lies in the ability to carry out all queries (in step 4) within a single UDP packet, which is more efficient.

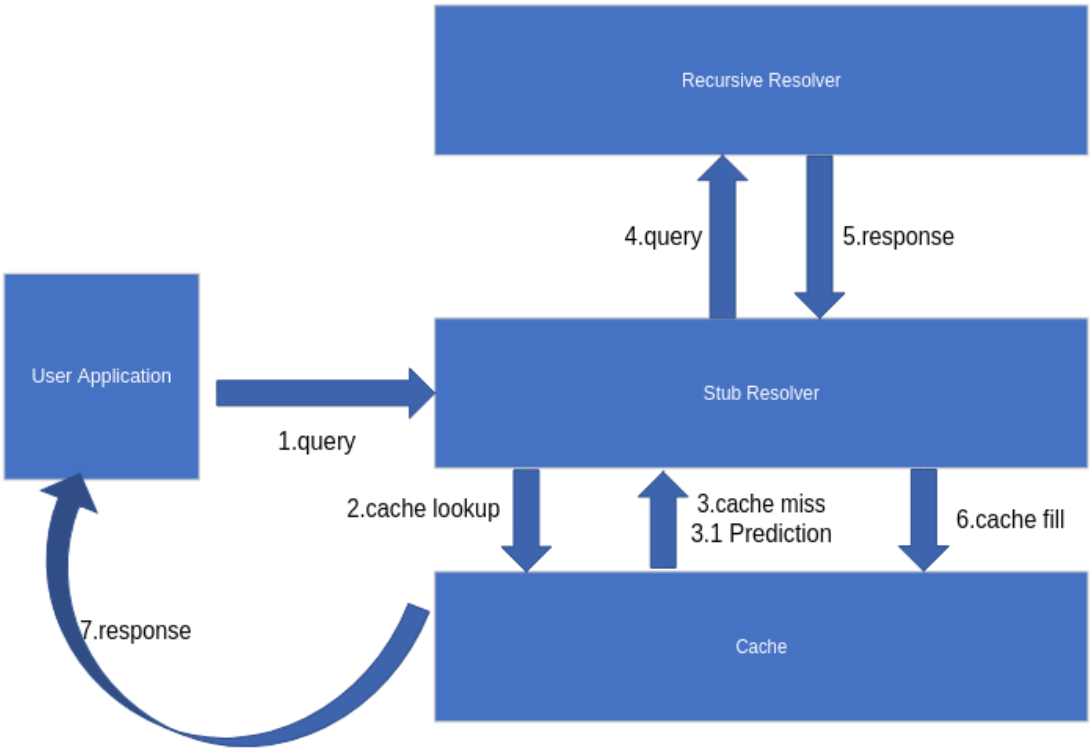

Figure 12: Integrated Prediction Application

The integrated prediction generates less traffic between stub resolver and recursive resolver, as shown in Figure 13.

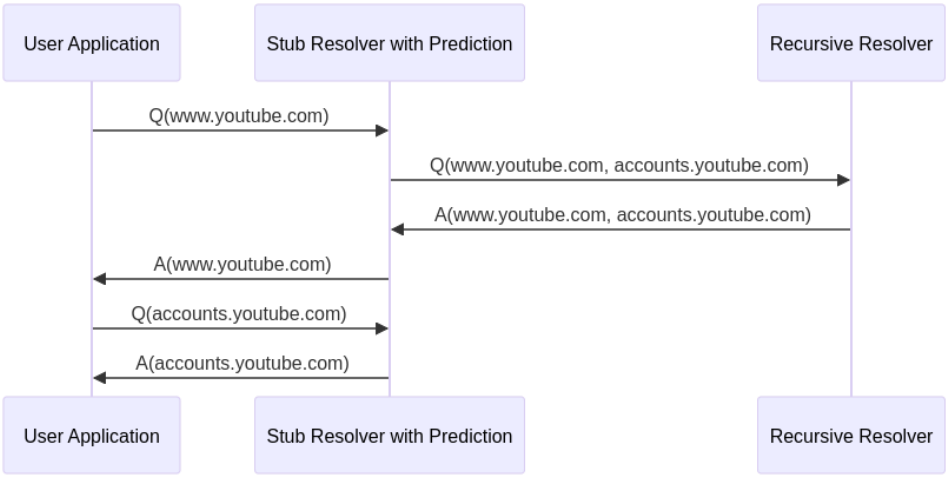

Figure 13: Sequence Diagram of Integrated Prediction

Improvement Estimation

The accurate figure of the improvement varies between hardware and software implementations. Experiment environment data on a Dell Inspiron 5593 laptop running Firefox as the web browser to load the home page of www.youtube.com shows in Figure (3).

	DNS Resolution (milli second)	Loading (second)	Queries Count
Max	5	5.7	17
Min	1	2.14	14
Typical	3	2.2	14

Figure 3: Experiment Environment Data

In Figure (3), typical time cost to load the web page is 2.2 seconds, 14 DNS queries are triggered in this loading process, which means 42 milli seconds among them are DNS resolution time cost. DNS contributes 1.9% of the time cost to this web application. Note other network applications may have different portions of time cost, considering that web applications spend most of their time parsing and rendering documents. Using the methods in section 3.3, the improvement estimation illustrated in Figure (4).

	Window=1000 Hops=1	Window=1000 Hops=2	Window=2000 Hops=1	Window=3000 Hops=1	Window=1000 Hops=3
DNS Reduce	15 ms	33 ms	21 ms	24 ms	33 ms
Time Ratio	33%	73%	46%	53%	73%
Cost	3	26	5	6	26

Scale Ratio	120%	273%	133%	140%	273%

Figure 4: Time Efficiency Improvement and Cost

In Figure (4), “DNS Reduce” is the time saved by prediction, which is directly determined by the precision rate. The “cost” is the number of DNS record entries scale up, which is affected by the recall rate. The “Time ratio” is the result of the reduced time divided by the time cost without prediction. The “Scale Ration” is the records number (including prediction) divided by the records number without prediction. To explain the calculation, take the first parameter set as an example, where window size is 1000 milliseconds and hops is one. The number of DNS queries involved is 15, the prediction has 5 correct and 3 wrong, the reduced time ration is 5/15, and the DNS cached record is scaled up from 15 to 18, with scale ratio 18/15.

To summarize, the learning methods and prediction applications proposed by this thesis trade more (2-3 times) memory, for less (30%-70%) DNS resolution time cost.

Reference:

1 Mockapetris P. RFC1034: Domain names-concepts and facilities.

2 Hoffman P. RFC 9364: DNS Security Extensions (DNSSEC).

3 Dickinson J, Dickinson S, Bellis R, Mankin A, Wessels D. RFC 7766: DNS transport over TCP-implementation requirements.

4 Hoffman P, McManus P. RFC 8484: DNS queries over https (DOH).

5 Hu Z, Zhu L, Heidemann J, Mankin A, Wessels D, Hoffman P. RFC 7858: Specification for DNS over transport layer security (TLS).

6 Dilley J, Maggs B, Parikh J, Prokop H, Sitaraman R, Weihl B. Globally distributed content delivery. IEEE Internet Computing. 2002.

7 Hong YS, No JH, Kim SY. DNS-based load balancing in distributed Web-server systems. The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06). 2006.

8 Peter L. DNS Prefetch. US20100146415A (Patent). 2010.

9 Mozilla Corp. Using DNS-Prefetch. Available from: https://developer.mozilla.org/en-US/docs/Web/Performance/dns-prefetch. 2023.

10 Cardellini V, Colajanni M, Yu PS. Request redirection algorithms for distributed web systems. IEEE transactions on parallel and distributed systems. 2003.

11 Slattery S, Craven M. Combining statistical and relational methods for learning in hypertext domains. International Conference on Inductive Logic Programming 1998 Jul 22 (pp. 38-52). Berlin, Heidelberg: Springer Berlin Heidelberg.

12 Batal I, Fradkin D, Harrison J, Moerchen F, Hauskrecht M. Mining recent temporal patterns for event detection in multivariate time series data. The 18th ACM SIGKDD international conference on Knowledge discovery and data mining. 2012.

13 Prakasa Rao B.L.S., Conditional independence, conditional mixing and conditional association. Annals of the Institute of Statistical Mathematics. 2007.

14 Albers K, Bodmann F, Slomka F. Hierarchical event streams and event dependency graphs: A new computational model for embedded real-time systems. 18th Euromicro Conference on Real-Time Systems (ECRTS'06). 2006.

15 Minaei-Bidgoli B, Lajevardi SB. Correlation mining between time series stream and event stream. Fourth International Conference on Networked Computing and Advanced Information Management. 2008.

16 Evert S. The statistics of word cooccurrences: word pairs and collocations.

17 Chaudhari D, Damani OP, Laxman S. Lexical co-occurrence, statistical significance, and word association. arXiv preprint arXiv:1008.5287. 2010 Aug 31.