# Determining DNS over QUIC Susceptibility to Fingerprinting Attacks

Aakash Patel
aakashdp@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Jay Schauer
jschauer@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

## ABSTRACT

Recent years have seen rapid development and adoption of encrypted DNS techniques, including DNS over TLS (DoT), DNS over HTTPS (DoH), and most recently, DNS over QUIC (DoQ). These techniques aim to protect user privacy by encrypting DNS queries and responses; however, previous studies have shown that traffic analysis attacks can still reasonably deduce the content for encrypted DoT and DoH traffic. In this study, we show that DoQ is also susceptible to traffic analysis attacks. We develop a machine learning classifier that infers DNS traffic based on patterns of packet size and timing. Our classifier achieves up to 98.4% accuracy on our dataset of 200 websites. We then discuss different mitigation strategies, and show that while padding can defend against traffic analysis attacks to a degree, disguising the timing information is more helpful in defense. Our results demonstrate that traffic analysis attacks remain a concern with DoQ, and highlight the importance of further research on effective strategies to mitigate such attacks.

## 1 INTRODUCTION

The Domain Name System (DNS) protocol is widely used to translate domain names to IP addresses, and is foundational to Internet communication. DNS communication is generally sent unencrypted in plaintext [6]. Though this aids in fast resolution time and low complexity, it poses a privacy and security concern since malicious actors can passively collect information about which websites a user is visiting, or also potentially spoof DNS lookups to redirect users to an unintended website.

To protect user privacy and prevent such attacks, many new techniques have been proposed to encrypt DNS communication. Some early attempts such as DNSCurve in 2010 [8] did not gain widespread adoption. However, the popularity of encrypted DNS has been increasing in recent years; the Internet Engineering Task Force (IETF) standardized DNS over TLS (DoT) in 2016 [15], DNS over HTTPS (DoH) in 2018 [12], and most recently, DNS over QUIC (DoQ) earlier this year [16].

The use of encrypted DNS has been on the rise; a recent study found that the number of open DoT resolvers in the US increased by 400% between February and May 2018, and that DoT traffic grew 54% between July and December of that year [19]. In 2019, Mozilla announced that DoH will be enabled by default in Firefox in the United States, and are continuing its rollout in other countries [2]. DoQ is still in its infancy, and usage is currently low. However, it already outperforms DoT and DoH, and usage will continue to grow over time [17].

Although encryption is the first step in ensuring network security and privacy of user information, *fingerprinting attacks* have been shown to be effective in inferring the content of the traffic using features such as timing, size of packets, and duration of flows [30, 31]. Recent research has shown that DoT and DoH are both susceptible to such attacks [13, 29], though due to its recency, no such study has been done on DoQ.

In this paper, we demonstrate that like DoT and DoH before it, DoQ is also susceptible to fingerprinting attacks. We consider the case of a passive adversary that can see all traffic between the client and the resolver. Using modern machine learning methods, we show that even simple timing and packet size information is sufficient to achieve high classification accuracy for a variety of websites.

We then evaluate defenses against such an attack, including padding and random delay. We find that these techniques offer some degree of protection, but are unable to completely mitigate traffic analysis without considerable added overhead or latency. Finally, we discuss limitations of our approach and directions for future work.

## 2 BACKGROUND

DNS resolution is the process of translating a domain name, such as `example.com`, into an IP address, such as `93.184.216.34`. This process begins with a DNS query sent by the client to a recursive resolver. The recursive resolver then conducts a sequence of queries that let it determine which authoritative nameserver holds the mapping from the desired domain name to its IP address. The recursive resolver then obtains the IP address from the authoritative nameserver and sends it back to the client, which can then connect to the desired host with the IP address [3].

This process generally occurs fairly quickly, and is made more efficient through the use of caching at all levels of the hierarchy - from the browser itself, the operating system, the recursive resolver, and the various servers that the recursive resolver communicates with.

## 2.1 DNS over TLS

DNS over TLS was one of the first encrypted DNS protocols to be standardized by IETF. DoT consists of the client first establishing a TLS connection with the recursive resolver over a “well-known port number" (typically TCP port 853) [15]. Since the port is dedicated to DoT traffic, this gives network administrators more control over DNS queries coming into or going out of their servers, but also makes it easier for an attacker to isolate the DNS traffic (though the packets themselves remain encrypted).

## 2.2 DNS over HTTPS

In contrast to DNS over TLS, DNS over HTTPS uses HTTP as the transport protocol instead of UDP. The DNS query is included in a HTTP request using the HTTP GET and POST methods.

HTTP headers offer more information that can be used for traffic analysis than those of TCP, such as the User-Agent and Authentication request fields [12]. However, whereas DoT has its own dedicated port number, DoH uses the standard HTTPS port 443. Since the DoH traffic looks indistinguishable from other HTTPS traffic, this provides an additional layer of privacy as an attacker would not be able to trivially differentiate the DNS traffic from other general HTTPS traffic.

## 2.3 DNS over QUIC

DNS over QUIC differs from DoT and DoH in that it is built over the QUIC transport protocol, instead of TCP or HTTP. QUIC has several advantages over TCP, including better speed, head-of-line blocking, connection migration, and built-in encryption [5]. UDP port 853 is reserved for DoQ [16].

Since HTTP is an application-layer protocol, DoH can also utilize QUIC as the underlying transport protocol with HTTP/3. However, the additional privacy concerns with using HTTP outlined in Section 2.2 make DoQ a more attractive option than using DoH with QUIC [23].

## 2.4 DNS Fingerprinting

Fingerprinting attacks are a form of passive analysis that aim to infer encrypted content based on features available to an eavesdropper, such as timing, packet size, and direction. DNS fingerprinting relies on the premise that accessing a website performs not only the DNS lookup for the website itself, but a number of subsequent resolutions for other domains such as ads, images, and other resources. For example, accessing `google.com` produces queries for the following domains, in order: (1) google.com, (2) gstatic.com, (3) apis.google.com, (4) play.google.com, and (5) adservice.google.com. Since websites load different resources and in different orders, each website produces an identifiable sequence of queries and responses, even when the data itself is encrypted.

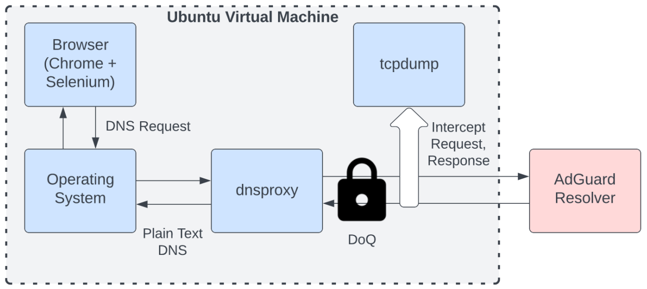

**Figure 1: Overview of data collection.**

Fingerprinting attacks with general web traffic have been well studied, and have been shown to be effective [25, 30, 31]. Furthermore, DNS fingerprinting has also been studied with DoT and DoH [7, 13, 25, 29]. Since QUIC does not fundamentally alter the sequence of queries and responses, we expect that DoQ should also be susceptible to fingerprinting attacks.

## 3 APPROACH

## 3.1 Data Collection

*3.1.1 Environment and Setup.* Our data collection was conducted on an Ubuntu 20.04 virtual machine on our university network. We use Selenium version 4.5 to automate visiting a website using headless Chrome version 107. The browser is launched in incognito mode and we also disable caching to ensure that client-side caching does not interfere with the data collection.

Chrome does not currently have native support for DoQ. Therefore, we use dnsproxy [4] to redirect DNS queries from the stub resolver to AdGuard's public DoQ resolver at `quic://dns.adguard.com`. We capture the trace for each sample using `tcpdump`, listening on port 853 to capture just the DoQ traffic. An illustration of this setup appears in Figure 1.

We collect traces for the top 200 sites in the Alexa top 1 million websites list [1]. However, many of these sites time out or are otherwise unaccessible. Therefore, we first manually curated this list by eliminating domains that did not yield a valid trace in two out of three trials, and then used the top 200 from the curated list for the final collection. Each website is visited in a round-robin fashion for a total of 100

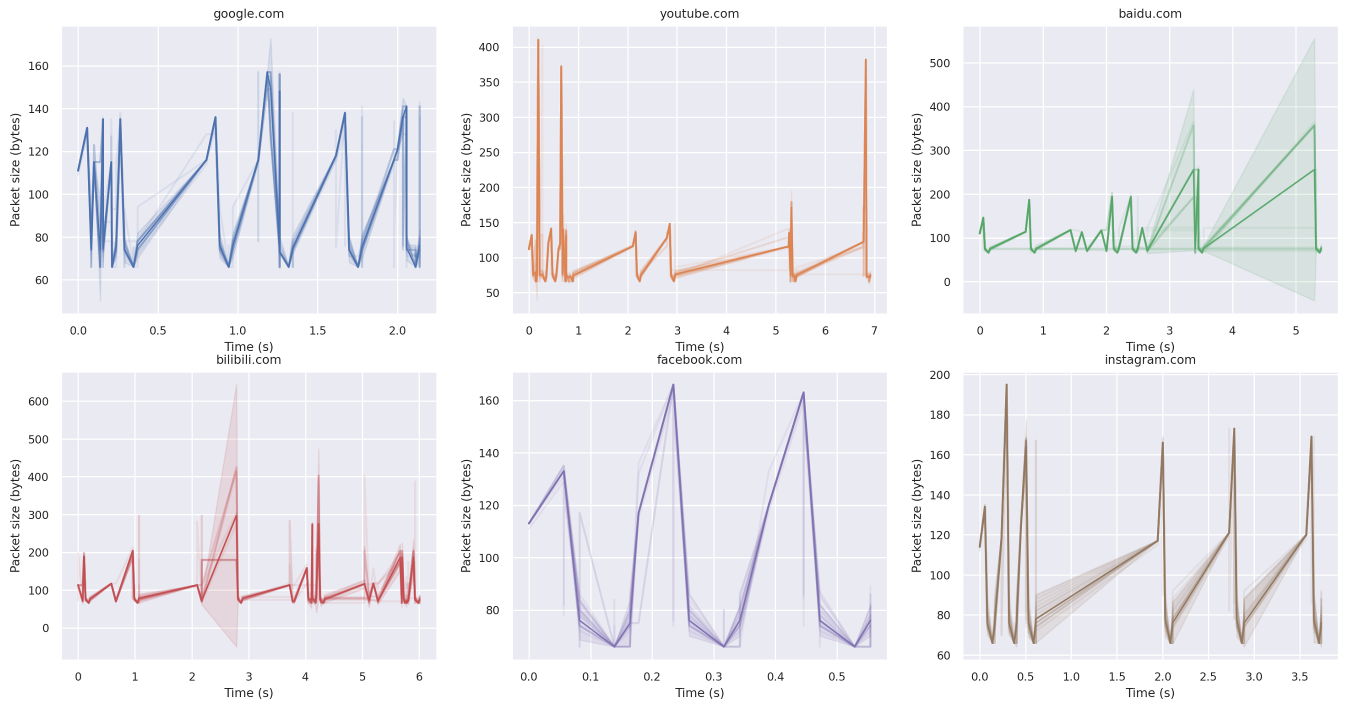

**Figure 2: Sample data for top six domains. The bold line is the median trace, with one median absolute deviation above and below it represented by the shaded region. The data shows clear patterns within a class and differences between classes, motivating the use of machine learning methods to classify the website based on the trace.**

iterations, with any invalid samples (due to time out or other exceptions) discarded. The data was collected throughout October and November 2022.

*3.1.2 Dataset.* Our final dataset consists of 19,788 total samples over 200 classes. The dataset is fairly balanced, with 163 classes having a full 100 samples, and a majority of the remaining classes at least 90 (the minimum is 72). The data consists of a total of 1,431,277 packets, for each of which we record the time stamp, packet size, and whether the packet is incoming or outgoing.

An example of some of our data for the top six sites in our list appears in Figure 2. For this visualization, the traces were aligned using dynamic time warping [28], and the median values are plotted as the bold line for each class. A more in-depth explanation of dynamic time warping and how this figure was generated is given in Appendix .

## 3.2 Feature Selection

Based on related work in this area [24, 26], we use a time series representation of the data with three features that an attacker would have access to: the time at which a packet arrives, the size of the packet, and the direction of the packet (determined by the source and destination IP addresses). Note that each time series may have a variable number of time points. We also use the time between packets (inter-arrival time) instead of the raw arrival time in some experiments.

## 3.3 Classifier Design

To determine our classifier design, we first experimented with a variety of general classification algorithms including linear regression and random forest, as well as specialized time series classification algorithms including Catch-22 [20] and ROCKET [9]. From our preliminary results, we found that ROCKET generally outperformed the other models, leading us to use it as the baseline in our later experiments.

ROCKET uses random convolutional kernels as a feature transformation. Traditionally, weights of convolutional kernels are learned during training. However, random convolutional kernels have shown to also be effective in other applications such as facial recognition and representation learning [9]. ROCKET leverages this to compute a large number (on the order of thousands) of random kernels, which are then applied to the input data.

Catch-22 is a subset of 22 features from the *hctsa* toolbox, a comprehensive MATLAB library of time series features. The selected features capture a diverse array of properties, and have been shown to have high performance over a variety of time series classification tasks [20].

We also run some experiments with MINIROCKET, a reformulated version of ROCKET that has similar accuracy, but is considerably faster [10]. We find that MINIROCKET actually has slightly better performance than ROCKET on our dataset, but the results are comparable.

The transformed data is scaled to have a zero mean and unit standard deviation, before feeding it to a linear estimator. We experimented with ridge regression, logistic regression, and support vector machines (SVM) as our estimators of choice.

We use the Python `sktime` library [18] for ROCKET, Catch-22, and MINIROCKET implementations, and the `scikit-learn` library [27] for other machine learning algorithms that we used.

## 3.4 Mitigation Strategies

Despite encrypting packet payloads, encrypted DNS has been shown to be vulnerable to fingerprinting attacks by leaking information through packet sizes and timing, as shown in [7, 14, 29]. Strategies to mitigate the effectiveness of fingerprinting techniques attempt to reduce the information available by making traffic appear more uniform. Two approaches can be taken in parallel: padding packet sizes, and modifying packet timing.

*3.4.1 Packet Size Padding.* RFC 8467 includes multiple strategies for padding DNS messages to hamper correlation methods [22]. The recommended method is block-length padding where queries are padded to the closest multiple of 128 bytes and responses are padded to the closest multiple of 468 bytes. These sizes were found to be a good trade-off between cost to attacker and cost to defender. They also discuss max-length padding, where all packets below a certain threshold are padded to that value. To implement these strategies, we take our existing samples and modify the recorded packet size before running each sample through the classifier. We then record modified accuracy and the overhead due to the additional bytes. However, just modifying packet sizes is not enough to prevent fingerprinting, timing information must also be modified, as shown in [7].

*3.4.2 Packet Timing.* Packet timing features such as inter-arrival time, overall transmission time, and query rate have been shown to be important for fingerprinting attacks against encrypted DNS [14, 21]. In order to mitigate the information provided by these features, several strategies can be employed. [7] evaluates a constant-rate (CR) mitigation, in which packets are sent at a fixed schedule, and if no DNS payload is waiting, then dummy packets are sent in place. While CR-mitigation eliminates most timing information,

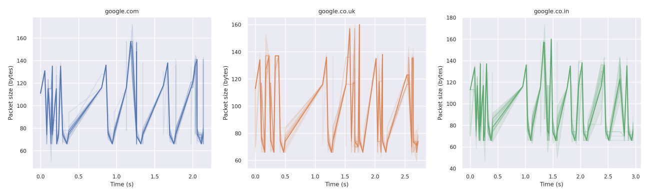

**Figure 3: Sample data for different google.* country-code domains. We consider these classes to be equivalent when evaluating the accuracy of our model.**

it imposes large bandwidth and latency overhead. In this paper we instead investigate adding a uniformly distributed random delay to each packet. We evaluate the effect of this delay against classifiers trained on samples with absolute time and inter-arrival time.

To implement the uniform random delay for absolute-time, we start by adding a random delay to the arrival time of the second packet in a sample. That delay is then propagated and added to the arrival time of all subsequent packets, and this process is repeated for each packet. To implement the delay when the classifier uses inter-arrival time, we simply add a uniform random amount to each packet's inter-arrival time. In both cases, we assume the client and server use the same uniform distribution. After the delay is added, we evaluate the adjusted samples with the classifier and record modified accuracy and overhead in terms of total transmission time.

## 3.5 Code and Resources

Our code is made publicly available at https://github.com/jayschauer/eecs589-patel-schauer.

## 4 EVALUATION

## 4.1 Evaluation Metrics

We evaluate the effectiveness of our model using three metrics: the accuracy, the F1-score, and a modified accuracy that combines certain class labels.

In our dataset, different country-code domains (e.g. `google.com` and `google.co.uk`) have different labels. Since these domains generate the same DNS trace (see Figure 3), it is likely our model may err on these classes. To minimize the effect of such errors, we use a "modified accuracy" metric that considers these classes equivalent when determining the accuracy of the model.

We consider two domains to be similar when they differ only in the country code, or when one domain redirects to another. In our dataset, the only cases of the former are various country codes for `google.*` and `amazon.*`, and the only case of the latter is `redd.it` redirecting to `reddit.com`.

We also evaluate the effectiveness of the models with the F1-score. The F1-score is the harmonic mean of the precision,

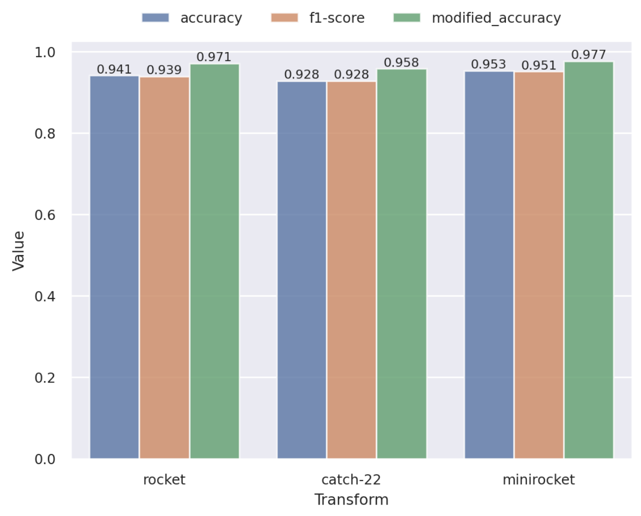

**Figure 4: Evaluation of different feature transforms.**

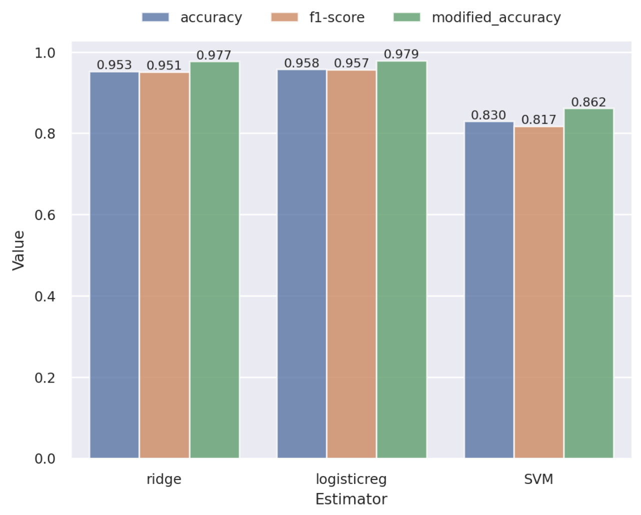

**Figure 5: Evaluation of different estimators.**

which is the ratio of true positives to the total number of positive predictions, and the recall, which is the ratio of true positives to the actual number of positives. The F1-score is generally a better metric to use than the accuracy when the data is imbalanced. Our data is well-balanced however, and as we will show, the F1-score and accuracy of all of our models are fairly similar, so we use our modified accuracy metric as our main method of evaluation.

We train our models using 75% of our dataset, and reserve 25% for evaluation.

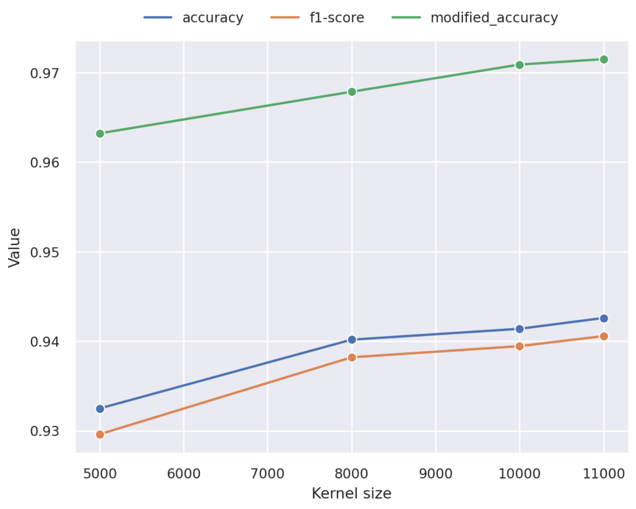

**Figure 6: Evaluation of number of ROCKET convolutional kernels.**

## 4.2 Classification Results

*4.2.1 Evaluating classifier design.* We first evaluate the effectiveness of different classifier design choices. We try three different feature transformations, using the ROCKET, Catch-22, and MINIROCKET transform followed by a ridge regression classifier. The results are displayed in Figure 4. MINIROCKET performs the best on all three metrics with a peak modified accuracy of 97.7%, though ROCKET and Catch-22 are not far behind. Notably, the accuracy and F1-scores are very close together for all three models, indicating that our model is not biased toward particular classes.

Next we consider the effectiveness of using different estimators - ridge regression, logistic regression, and support vector machine - following the MINIROCKET transform. The results appear in Figure 5. The ridge regression case is the same as the MINIROCKET case from Figure 4, and is used as a baseline. We find that logistic regression has a small gain over ridge regression, with a modified accuracy of 97.9%. Again the accuracy and F1-scores are almost equal for each model.

One major advantage of ROCKET is the ability to use a large number of kernels, which may be computationally infeasible with other models. Therefore, we also investigate the number of random kernels used with ROCKET as a model hyperparameter. We experiment with 5,000, 8,000, 10,000, and 11,000 kernels. The results appear in Figure 6. As expected, the model performance increases with an increasing number of kernels, though the gain between 8,000 and 11,000 is much smaller than the gain between 5,000 and 10,000. As such, for the remaining experiments we use the default value of 10,000 kernels, as it provided a good trade-off between performance and training time.

*4.2.2 Evaluating feature importance.* Next, we evaluate the importance of different features by training the ROCKET transform with ridge estimator using a subset of the dataset's features and comparing model performance. This is important because it informs our choice of mitigation strategy.

Each sample in the dataset contains time, size, and direction information. Direction is a binary class: 1 for outgoing and -1 for incoming packets. 4 classes of tests were run: (1) including direction information for each packet, (2) only using outgoing packets, (3) only using incoming packets, and (4) using all packets but including no direction information. These tests are labeled 'both', 'outgoing', 'incoming', and 'none', respectively. Within each class, four feature subsets were used: both time and size information, just time, just size, and just direction. The results can be seen in Figure 7. These experiments can be interpreted as evaluating 'perfect mitigations' as described in [7], where a perfect mitigation is one which removes all available information from a given feature. In other words, perfect padding would mean no useful size information is available, only timing information, and perfect timing defense would remove all useful timing information, leaving only size.

First, we see that using time and size together results in better modified accuracy than just time or size alone. Also, size alone performs better than time alone. Surprisingly though, including direction information resulted in lower accuracy than the model trained without direction information. This perhaps is caused by the model overfitting on the training set. We see too that training using only direction information results in reduced performance, and conclude that direction is not as useful a feature for our classifier.

We can also conclude that using both time and size mitigations are necessary in order to fight fingerprinting. A defense that only uses either perfect padding or perfect timing defense would still be susceptible to a classifier trained on the feature that isn't protected.

*4.2.3 Raw arrival times vs inter-arrival times.* Thus far, we have used the raw timestamp of the packet arrival time as a feature for our model. However, the time between packets ("inter-arrival time"), has been shown to also be a useful feature [11, 13]. We compare two models where one uses the raw times, and the other uses the time between two consecutive packets (regardless of direction). Both models use MINIROCKET and a logistic regression classifier.

The results appear in Table 1. The inter-arrival time has a small increase in performance, likely due to it being more robust to small jitters in the network latency. We discuss the effectiveness of raw times and inter-arrival times further in Section 4.3.2.

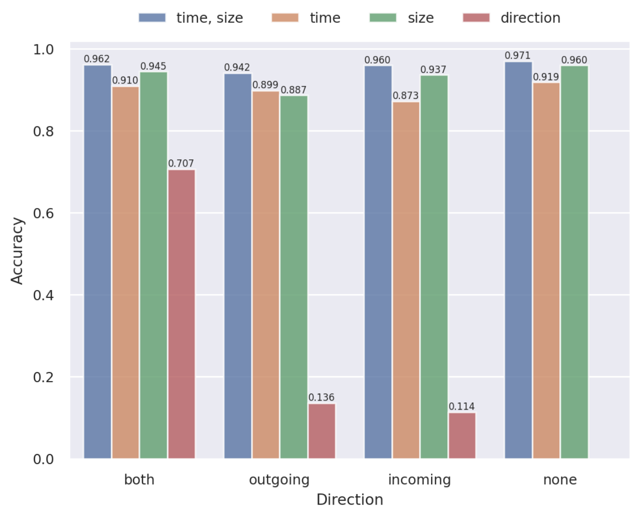

**Figure 7: Evaluation of different feature subsets. The X-axis shows which direction information was included during training and testing. Size seems most important, followed by time and then direction.**

**Table 1: Raw time vs inter-arrival time**

| Feature | F1-Score | Modified Accuracy |
|---|---|---|
| Raw time | 0.957 | 0.979 |
| Inter-arrival time | 0.961 | 0.984 |

*4.2.4 Evaluating sample length effect on performance.* Finally, we evaluate sample length's effect on performance. Here we use the ROCKET transformer with ridge estimator again. The median sample contains 56 packets, while the longest sample contains 374. We evaluate two cases: (1) classifier accuracy over all sample slices limited to a maximum length, with the start of the slice at regular increments within each sample, and (2) only measuring accuracy on the first N packets from a truncated sample. In case (1), slices start at increments of $length/2$ and end after $length$ samples or at the last packet. We adjust each sample according to these methods and test them with the classifier. In the second column of Table 2, we see that accuracy quickly decreases as maximum sample length decreases. However, if the beginning of a sample is still captured and only the end of a sample is truncated, then accuracy decreases more slowly, as shown by the third column.

## 4.3 Mitigation Results

*4.3.1 Padding Mitigation.* We evaluate the padding mitigation strategies outlined in Section 3.4.1 using the MINIROCKET transformer with a logistic regression estimator. Results are

**Table 2: Sample Length versus Modified Accuracy.**

| Length | Overall Accuracy | Truncated Accuracy |
|---|---|---|
| 374 | 0.956 | 0.956 |
| 200 | 0.762 | 0.948 |
| 100 | 0.465 | 0.804 |
| 50 | 0.216 | 0.596 |
| 25 | 0.062 | 0.301 |
| 10 | 0.040 | 0.043 |

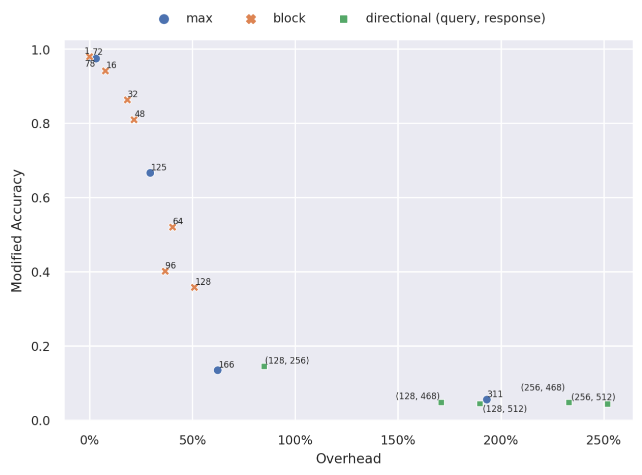

**Figure 8: Scatter plot showing padding mitigation strategy accuracy versus overhead.**

shown in Figure 8. We see that both block and max padding strategies work well to decrease the classifier accuracy. Under our classifier, the RFC 8467 strategy appears to be overkill, since compared to strategies with less overhead it only results in a small additional decrease in accuracy (down to 4.8%). Max-padding with a length of 166 bytes has the best trade-off in terms of decrease in accuracy versus overhead: 11.7% accuracy versus 61.5% overhead. 166 bytes is the 90th percentile of packet size in our dataset.

*4.3.2 Delay Mitigation.* We evaluate the uniform random packet delay strategy using the MINIROCKET transform with 2 estimators - ridge regression and logistic regression. Delays range from 1 millisecond to 1 second. We also test the effect of modeling time as absolute-arrival time versus inter-arrival time for each packet. Our samples have a median total time of 3.19s and median inter-arrival time of 0.026s. As seen in Figure 9, the models trained on absolute time perform very poorly under the delay mitigation, with modified accuracy decreasing to 0.08 even with a 1 millisecond delay. But we see that instead using packet inter-arrival time as the time feature in the model makes it much more robust.

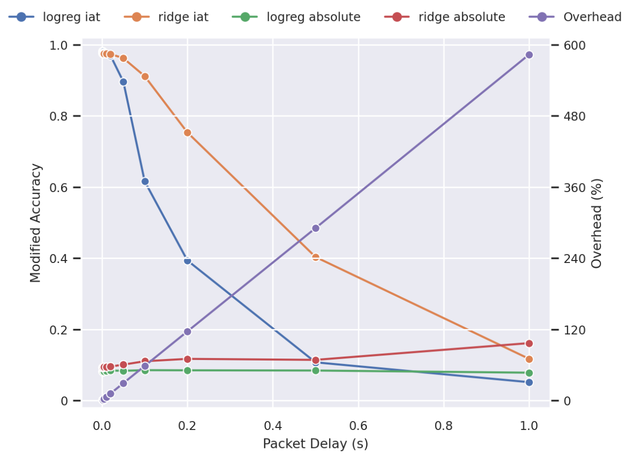

**Figure 9: As the random delay applied to packets increases, model performance decreases.**

The ridge estimator also handles the delay mitigation better than logistic regression, and we see a degradation in performance around 0.1s-0.2s of delay, with overall transmission time overhead of 58% and 116% respectively. The uniform random delay mitigation results in a higher overhead than the constant rate scheme in [7], but does not incur additional bandwidth usage.

## 5 DISCUSSION

Our experiments operate under several assumptions that would not hold true under real-world conditions. In practice, the fingerprinting accuracy would be lower due to additional noise and variation in DNS queries. Thus what we found can be seen as an upper limit to what is possible by a real-world attacker.

### 5.1 Threat Model Limitations

We assume an eavesdropper is listening to DNS queries somewhere on the network between the client and resolver. This adversary could be someone like the client's ISP. This poses several challenges compared to what we evaluated in the paper: first, the adversary might not be able to easily distinguish when the DNS queries for a given website start and end. As shown in Section 4.2.4, capturing the full trace of DNS requests for a website visit, including the start of the trace, is important for accurate fingerprinting. Second, with multiple users from a home network accessing the internet simultaneously, it could be difficult for the attacker to separate DNS requests from different individual users. If DNS requests for different websites are co-mingled, then it might become impossible to determine what website was visited. Second, we assume that the user is only visiting one of 200 websites, and so the attacker only needs to determine which of these 200 websites is visited. In reality, many more websites would need to be included in the training data, and there would still remain the possibility that a user visits a website that wasn't included.

### 5.2 Data Collection Limitations

There are several differences between our method of data collection and what would happen in the real-world that reduce the noisiness of our data. First, all our samples are collected from the same server, using the same browser, DNS client, and DNS resolver. A real attacker would experience many more combinations of these parameters, which could change packet timing or order. Second, websites can change over time, and so their DNS fingerprint might not be constant. We did not evaluate the performance of our classifier on data collected from a different time than when it was trained.

Third, a user might employ an ad-blocker, which would also change the DNS fingerprint, by blocking ad-serving domains from showing up in the trace. Fourth, we disabled caching while collecting data. If a user caches DNS requests, that could prevent many requests from appearing to the attacker, and decrease the amount of information the attacker has access to. However, [14] discusses caching in more detail and find that this problem could likely be overcome in practice.

Fifth, we assume that users only visit the home page for each domain in our dataset. Different pages under a given domain could have different DNS traces. We also assume the user only visits one website at a time. In practice, a user doesn't have this limitation and might open multiple tabs in the background simultaneously, which would prevent a classifier trained on single-page visits from determining what the user actually loaded.

Finally, our method of data-collection cannot detect when a website has truly finished loading to stop the trace for that website. The Selenium webdriver (which we used to visit websites programmatically) considers a website loaded once the *document.readyState* is *complete*. However, it is still possible for a website to load resources later on due to javascript executing.

### 5.3 Implications of Limitations

Due to the many confounding factors that would reduce the performance of a general DNS fingerprint classifier, an attacker might have difficulty targeting a general population with fingerprinting. So an important future work would be to perform fingerprinting using traces of real encrypted DNS traffic. Such a dataset is difficult to obtain because it might not be possible to retroactively determine what website corresponds to the DNS requests, which would be necessary for supervised training. Collecting data on realistic DoQ traffic would be very useful, as it could be used to evaluate the true feasibility of DNS fingerprinting and the relevancy of our threat model.

## 6 RELATED WORK

Previous studies have shown that DoT and DoH are susceptible to traffic analysis attacks [7, 13, 25, 29].

Houser et al [13] develop a method similar to ours for fingerprinting DoT traffic. Whereas we use the ROCKET transform to extract features from the time series data, they use summary-based features such as query and response length, total number of queries and responses, time intervals, queries per second, and time to receive the first N bytes. They use two ensemble methods - random forest and Adaboost - as their classifiers of choice. They report 93% accuracy on unpadded data, and 78.7% with padded data. However, they do not investigate other padding methods, or other defenses such as time delay.

Siby et al [29] analyze DoH traffic, using a novel feature set of n-grams of TLS record lengths. The trace is represented as a sequence of integers with the sign indicating the direction and the absolute value indicating the size. Similar to Houser, they also use a random forest classifier. This method does not use timing information as a feature, but they are still able to achieve an F1-score of 0.934 with combined labels (similar to our modified accuracy). They also find that their model still achieves an F1-score of 0.43 with the recommended EDNS(0) padding, similar to our results.

Bushart et al [7] analyze the effects of different padding strategies on encrypted DoH and DoT traffic. They achieve 86.1% accuracy on padded data using a k-Nearest Neighbors classifier with a sequence of message size and time as the features. Importantly, they show that their method is also effective for identifying subdomains. Similar to us, they evaluate "perfect padding" and "perfect timing" strategies, and find that perfect timing is a much more effective defense than perfect padding.

## 7 CONCLUSION

Our work demonstrates that DoQ suffers from similar privacy concerns as other encrypted DNS protocols, and underlines the importance of keeping data privacy in mind as the DoQ protocol continues to be developed and implemented. We show that packet size and timing information can be used to infer the content of encrypted DNS traces with high accuracy. Our analysis of padding strategies demonstrates that padding is able to mitigate traffic analysis to a certain degree, but cannot completely prevent it without significant overhead. We find that disguising timing information may provide more protection than just padding. We hope that our findings help guide future research on DNS over QUIC privacy and the development of more effective methods to mitigate traffic analysis attacks on DoQ.

## REFERENCES

[1] 2022. Alexa Top Websites. https://www.expireddomains.net/alexa-top-websites/

[2] 2022. Firefox DNS-over-HTTPS. https://support.mozilla.org/en-US/kb/firefox-dns-over-https

[3] 2022. What is DNS? | how DNS works. https://www.cloudflare.com/learning/dns/what-is-dns/

[4] AdguardTeam. 2022. AdguardTeam/dnsproxy: Simple DNS proxy with Doh, Dot, Doq and DNSCrypt support. https://github.com/AdguardTeam/dnsproxy

[5] Vasily Bagirov. 2020. Adguard DNS-over-QUIC. https://adguard.com/en/blog/dns-over-quic.html#instruction

[6] S. Bortzmeyer. 1970. DNS privacy considerations. https://www.rfc-editor.org/rfc/rfc7626#section-2.1

[7] Jonas Bushart and Christian Rossow. 2020. Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*. USENIX Association. https://www.usenix.org/conference/foci20/presentation/bushart

[8] Matthew Dempsky. 2010. DNSCurve: Link-level security for the Domain Name System. https://datatracker.ietf.org/doc/html/draft-dempsky-dnscurve-01

[9] Angus Dempster, François Petitjean, and Geoffrey I Webb. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery* 34, 5 (2020), 1454–1495.

[10] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. 2021. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 248–257.

[11] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*. 1187–1203.

[12] P. Hoffman and P. McManus. 1970. DNS queries over HTTPS (DOH). https://www.rfc-editor.org/rfc/rfc8484

[13] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang. 2019. An investigation on information leakage of DNS over TLS. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 123–137.

[14] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang. 2019. An Investigation on Information Leakage of DNS over TLS. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies* (Orlando, Florida) *(CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 123–137. https://doi.org/10.1145/3359989.3365429

[15] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. 1970. Specification for DNS over Transport Layer Security (TLS). https://www.rfc-editor.org/rfc/rfc7858.html

[16] Christian Huitema, Sara Dickinson, and Allison Mankin. 2022. DNS over Dedicated QUIC Connections. https://www.rfc-editor.org/rfc/rfc9250

[17] Mike Kosek, Trinh Viet Doan, Malte Granderath, and Vaibhav Bajpai. 2022. One to Rule Them All? A First Look at DNS over QUIC. In *International Conference on Passive and Active Network Measurement*. Springer, 537–551.

[18] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz J Király. 2019. sktime: A unified interface for machine learning with time series. *arXiv preprint arXiv:1909.07872* (2019).

[19] Chaoyi Lu, Baojun Liu, Zhou Li, Shuang Hao, Haixin Duan, Mingming Zhang, Chunying Leng, Ying Liu, Zaifeng Zhang, and Jianping Wu. 2019. An end-to-end, large-scale measurement of dns-over-encryption: How far have we come?. In *Proceedings of the Internet Measurement Conference*. 22–35.

[20] Carl H Lubba, Sarab S Sethi, Philip Knaute, Simon R Schultz, Ben D Fulcher, and Nick S Jones. 2019. catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery* 33, 6 (2019), 1821–1852.

[21] Minzhao Lyu, Hassan Habibi Gharakheili, and Vijay Sivaraman. 2022. A Survey on DNS Encryption: Current Development, Malware Misuse, and Inference Techniques. *ACM Comput. Surv.* (jul 2022). https://doi.org/10.1145/3547331 Just Accepted.

[22] Alexander Mayrhofer. 2018. Padding Policies for Extension Mechanisms for DNS (EDNS(0)). RFC 8467. https://doi.org/10.17487/RFC8467

[23] Andrey Meshkov. 2022. DNS-over-QUIC becomes proposed standard: Why it is good news for your privacy. https://www.techradar.com/opinion/dns-over-quic-becomes-proposed-standard-why-it-is-good-news-for-your-privacy

[24] Mohammadreza MontazeriShatoori, Logan Davidson, Gurdip Kaur, and Arash Habibi Lashkari. 2020. Detection of doh tunnels using time-series classification of encrypted traffic. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 63–70.

[25] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale.. In *NDSS*.

[26] Constantinos Patsakis, Fran Casino, and Vasilios Katos. 2020. Encrypted and covert DNS queries for botnets: Challenges and countermeasures. *Computers & Security* 88 (2020), 101614.

[27] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[28] Pavel Senin. 2008. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA* 855, 1-23 (2008), 40.

[29] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. 2019. Encrypted DNS–> Privacy? A traffic analysis perspective. *arXiv preprint arXiv:1906.09682* (2019).

[30] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1928–1943.

[31] Tao Wang and Ian Goldberg. 2016. On Realistically Attacking Tor with Website Fingerprinting. *Proc. Priv. Enhancing Technol.* 2016, 4 (2016), 21–36.

## A DYNAMIC TIME WARPING

Dynamic time warping (DTW) is an algorithm used to compute the similarity between two time series, even when the time periods of the two series differ. It is frequently used for speech recognition, since fluctuations in speaking speed can otherwise greatly impact the ability to match two spoken traces [28].

Dynamic time warping is an optimization problem that finds an optimal mapping between two time series where the cost, the difference between the values at two points, is minimized under certain constraints. Based on this mapping, one series can be "warped" to have the same time scale as the other, and the values at each time point can be compared directly.

Though DNS traces for a given website are generally quite similar, there are often small differences in time points due to network dynamics, and occasionally the number of packets differs between samples, possibly due to our limitation on knowing when a page is done loading described in Section 5.2. This makes dynamic time warping an attractive option as a feature transformation to reduce the noise between samples within a class.

In practice, we found that a k-nearest neighbors classifier using the distance from dynamic time warping as the distance metric performed much worse than ROCKET and other methods. This may be because DNS lookups are generally short, and often involve similar static resources (such as from `gstatic.com`) causing the cost difference between certain samples to be low. Because of this, we did not consider DTW-based models further for classification. However, DTW does lend itself quite well to visualizing our data, as we show in Figures 2 and 3.

These figures were generated by taking 20 samples from each class. For each class, one sample was selected to be the "candidate trace", and the remaining 19 samples were aligned with the candidate using dynamic time warping. We then compute the median at each time point on the aligned traces, as well as the median absolute deviation. The median absolute deviation (MAD) is defined as the median of the individual deviations from the overall median at each time point:

$$MAD = \text{median}\left(|X_i - \text{median}(X)|\right) \tag{1}$$

In these figures, we plot the individual traces as the light-colored lines, and the median value at each time point as the bold line. One median absolute deviation above and below the median is represented by the shaded region, showing the variability at each time point. We chose to use the MAD instead of the more common standard deviation since the MAD is more robust to outliers than the standard deviation.