CS 2630 Final Project: Adaptive Certificate Prefetching for Secure Web Protocols

Madison Davis
Harvard College
madisondavis@college.harvard.edu

Rosa Wu
Harvard College
rosawu@college.harvard.edu

Christine Tian
Harvard College
christinetian@college.harvard.edu

Abstract

Online Certificate Status Protocol (OCSP) stapling is a mechanism that allows web servers to prefetch certificate statuses from CAs to reduce client-side latency. However, existing stapling systems are prone to soft-fails, where clients may fail to receive an OCSP response but proceed with a TLS connection regardless. This can be due to web-server or CA server latency, high traffic, or system-wide power outages. Ultimately, soft-fails degrade security and partially negate the benefits of stapling. In this work, we propose a dynamic, CA-aware prefetching algorithm that incorporates (1) CA-to-server push telemetry reporting real-time latency, variance, and stability, and (2) adaptive prefetching that adjusts refresh timing based on CA performance and web traffic conditions. We implement and evaluate three prefetching scheduling algorithms within a simulated distributed environment that includes clients, web-servers, CA servers, and load balancers. Experiments under diverse traffic patterns and CA load scenarios demonstrate that, under more constrained system settings, our dynamic algorithm reduces soft-fail rates while balancing the prefetch load more effectively than other scheduling algorithms. Our results therefore offer a viable alternative to strict must-staple policies without introducing excessive server overhead.

Keywords

OCSP, TLS, CA signing, prefetchig

1 Introduction and Motivation

Transport Layer Security (TLS) is a network protocol that allows for secure network traffic over HTTP connections. Part of the TLS handshake relies heavily on the presence of a certificate, which serves as a binding between a public key and a principal. That is, it provides confirmation to the client that they are talking with the expected server and not a malicious one. Certificates need to be issued by a trusted certificate authority (CA) and must be renewed regularly to ensure the associated keys remain fresh and subsequent security compromises can be mitigated. This introduces the concept of a revoked, or outdated, status. It is thus the responsibility of the CA to communicate when these certificate revocations occur, as well as in the client's best interest to ensure it is always in possession of an up to date certificate. While this infrastructure enables secure connection, the reliance on this additional CA often causes operational challenges as traffic scales.

Online Certificate Status Protocol (OCSP) stapling is a commonly-used mechanism utilized in TLS handshakes to provide the revocation status of a certificate. It's designed to overcome the drawback of standard OCSP, where the client queries synchronously for a certificate's revocation status from the CA directly. This approach adds latency to the process due to the waiting period between the client's request and the CA's response. OCSP stapling aims to shift the burden of this communication to the web server by having the web server query the CA in advance of a client TLS handshake to check the revocation status of a certificate. This response is cached and has its own timestamp after which it will need to be renewed. With this approach, whenever a client request is made, the server can simply check its previously cached result, provided that the timestamp is still valid, rather than having to initiate another query to the CA. This stapling approach thus reduces the latency of the client to CA communication.

However, even with OCSP stapling, systems are still faced with the consequences of soft-fails. A soft-fail is generally defined as thus: if a client is able to receive a fresh OCSP response within a certain time frame, it proceeds as normal. But even if the response does not return with that time frame, the client will still proceed as normal, simply assuming the initiated connection will be safe. Soft-fails are a motivating concern for the following reasons:

(1) **Latency Increases**: The client may not receive a certificate if the stapled message has gone missing, has a stale timestamp, or is invalid, in which case the browser may again try to query for the OCSP information itself. This inevitably reverts back the approach of standard OCSP.
(2) **Certificate Growth**: Statistical analysis of the web has approximated the number of circulating certificates to over 305 million entities [1]. In addition, sites such as LetsEncrypt issue over 340,000 certificates per hour [7]. With such unprecedented growth, stapling will become more burdensome.
(3) **CA Traffic Congestion**: Traditionally, CAs operate passively in OCSP stapling. While web servers shoulder the responsibility of deciding when to refresh their stapled responses, CAs simply answer requests. Some existing systems, such as Cloudflare's fixed-time refresh, rely on simplistic refresh heuristics that assume stable CA responsiveness. However, real-world CA responsiveness is heavily influenced by global demand and can suffer from abrupt latency spikes, uneven traffic patterns, and transient outages. These events increase the likelihood of receiving stale OCSP responses, causing clients to soft-fail and accept outdated certificate validity information, which is an undesirable yet common fallback in today's web ecosystem.

It is noteworthy that there have been efforts to curtail dynamic CA traffic congestion, but with unintended consequences on other parts of the OCSP stapling pipeline. To illustrate an example, TLS certificates hosted by CAs have been agreed upon under the TLS Baseline

Requirements to last at maximum 398 days, whereas cached certificates can exist within the span of a few hours to a few days. However, subsequent ballot casts by the CA/Browser Forum have voted to reduce the duration of certificates to a mere 49 days post March 2026 [5]. The reasoning for the change was to lessen the burden on CA certificate revocation: by reducing the amount of time a certificate is valid, there is a higher likelihood of the certificate expiring in its lifetime due to natural causes compared to if it became compromised and needed to be revoked. While the ballot move does indeed aid in revocation, it does not intuitively help prefetching: as certificates become shorter-lived, OCSP responses must be refreshed more frequently, putting additional burden on web-servers to deliver timely and accurate stapled messages to clients.

Security Concerns Soft-fails are inherently dangerous because when the client assumes that an outdated certificate is still valid, there are opportunities for attackers to take advantage. For example, when revocation information is blocked, an attacker can issue a MiTM attack by holding a compromised and revoked certificate to effectively cause the client to replace the outdated certificate with a faulty one [2]. In a similar vein, suppose an attacker can find a server's private key through a side-channel leak or server compromise when under pressure. Then, if the client fails-open (analogous to soft-fail but logs the error) and is willing to accept an indeterminate revocation status, the attacker can spawn up their own-controlled server, provide the client the revoked certificate, and the client would accept since it could not see the revocation status [3].

2 Problem Statement

In summary, there are several problems that arise in OCSP Stapling that motivated the creation of this paper:

(1) A missing or invalid certificate forces the system to fall back to the high-latency OCSP approach that stapling aimed to circumvent in the first place.
(2) There has been an increase in burden shifted to CA servers due to the number of new certificates being made and the need to service direct requests. Therefore, if the traffic of the CA server is high at the time of querying for the stale message, it will be difficult to complete in a timely manner.
(3) The client's assumption that the certificate is still valid leads to looser security guarantees and makes the system more vulnerable to attacks.

The goal of this paper is thus to improve upon OCSP stapling by reducing the chances of a soft-fail. We do this by introducing a dynamic prefetching scheduling algorithm, along with two additional fundamental prefetching algorithms for comparison. The introduction of any prefetching algorithm requires a delicate balance between receiving a successfully updated certificate from the CA and not overloading the CA with too many inefficient prefetches (for instance, if the CA would have been stable at the normal time of updating the certificate, then a prefetch now would be inefficient). We explore these nuances by examining how each algorithm would perform (soft-fail rate and prefetch efficiency) under various traffic conditions when used by a web-server. By addressing these concerns, we hope to improve both the latency and security of the TLS handshake without requiring hefty server configuration or impractical protocols, such as OCSP-must staple [4].

2.1 Overview of Prior Work

Prior mechanisms have been proposed to reduce certificate system latency, improve reliability, and avoid overloading certificate authority (CA) infrastructure. However, all fall short related to realistic, dynamic implementations.

Stricter Stapling OCSP must-staple is a stricter version of OCSP stapling. Whereas OCSP stapling would allow a client to request a direct OCSP check if there exists a soft-fail, OCSP must-staple mandates that the server must initially provide a valid staple, otherwise the client will revert to a hard fail. While this mechanism in theory would eliminate soft-fails entirely, it would require the current state of the web (incorporating all worldwide CAs, web browsers, and servers) to support OCSP must-staple, which at the time of writing is argued to be too strict for an actual implementation [4].

Static Refresh Other methods have tried instead to improve the *physical* systems involved in OCSP stapling. For instance, Cloudflare improves upon OCSP stapling by refreshing a certificate a fixed amount of time before it expires, then placing that refreshed value in a cache on the web server [8]. In this setup, the web server nor client need not have to go all the way to the CA server to query if the certificate expires, and they only need to check its closest respective cache. While a great practical implementation, Cloudfare mentions no implementation that involves a dynamic, adaptive policy which considers both (1) real-time and future-predicted traffic conditions to a CA server and (2) potential redundancy through CA load-balancing, as done in modern-day distributed systems. Indeed, in the aforementioned cited experiment, Cloudfare only displays their work for one CA server.

The Need for Dynamic Prefetching One notable question to address is the following: is dynamic prefetching necessary? We later explore this further across a variety of experimental setups, but prior work aids to shed light on this topic. In particular, a European study done by NetCraft discovered that requests to their servers were declined 6% of the time due to server overload [6]. In a similar vein, Cloudflare found that issues related to load balancers would cause 1 in 6 responses to come back as failures in their system [8]. There is therefore an urgency to curtail unnecessary server requests and improve response reliability, which can be done by dynamic pre-fetching.

2.2 Novelty and Research Contribution

Taking inspiration from the aforementioned prior work, this paper introduces two key novelties to this design space:

(1) **CA-to-server push-based telemetry**
We augment the traditional OCSP stapling workflow by enabling CAs to send periodic push notifications to web servers reporting their current mean response latency, latency variance, traffic volume, and an overall stability indicator. This transforms the server's refresh decision from a

passive, expiration-driven task to one informed by real-time CA load conditions.

(2) **Dynamic prefetching algorithms**
In parallel, our research contributes a performance evaluation of certificate refresh strategies across multiple traffic, expiration, and CA architecture regimes. We benchmark three algorithms (baseline, greedy, dynamic) across 100-node distributed simulations and quantify their impact on soft-fail rates and prefetch efficiency. This work is the first to integrate CA-side telemetry into OCSP stapling refresh scheduling and to empirically evaluate algorithmic refresh policies under controlled, multi-factor network conditions.

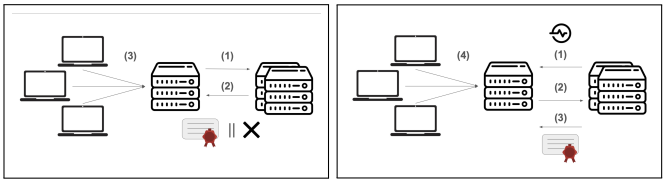


Figure 1: Side-by-side comparison of traditional OCSP stapling (left) vs our prefetching scheduler algorithm (right). On the left, (1) instantaites a prefetch request, (2) responds with a certificate or error due to CA overload, and (3) simulates client connections. On the right, we add an additional push notification by the CA that the web-server takes into account when requesting the CA server in (2).

3 Algorithm Technical Description

We implemented and evaluated three certificate prefetch scheduling algorithms within our simulated distributed system. Each algorithm embodies a different approach regarding how aggressively a web server should refresh its OCSP staple prior to certificate expiration. As a contextual note, due to the constraint that real-life cached certificates live for hours or days, in our experiment, we reduce the window of experimentation by assuming certificates are set to renew after 7 seconds of instantiation.

Baseline: The baseline algorithm provides a deterministic, fixed-time prefetching policy. It initiates a renewal request whenever the certificate enters a predefined prefetch window, which is set to 2 seconds before expiration in our implementation. This reflects common industry practice, where web servers refresh certificates at a fixed offset from their expiration time. While efficient under ideal conditions, this strategy is susceptible to CA overload or latency spikes occurring exactly during the refresh window.

Greedy: The greedy algorithm represents the opposite extreme. The server attempts to refresh its certificate at every scheduling interval (every second in our system). This eliminates nearly all risk of stale certificates but does so at the cost of excessive CA load and extremely poor prefetch efficiency. Greedy acts as a lower-bound comparison point to quantify the minimum possible stale rate if efficiency is ignored.

Dynamic: The dynamic algorithm is stability-aware and leverages the novel CA push-notification mechanism. Each CA sends its measured mean latency, latency variance, and a boolean stability flag to the web server every 0.5 seconds. The server classifies the CA as stable or unstable using thresholds on mean latency and variance. If the CA is stable, the server delays prefetching until it enters a smaller risk window (1 second before expiration) to avoid unnecessary load. If the CA is unstable, which is characterized by high variance or high average latency, the server adopts a larger "safe window" (2 seconds before expiration) and refreshes earlier, reducing the chance of getting stuck behind CA congestion during the critical expiration period.

Together, these algorithms span the spectrum from static to predictive to congestion-aware scheduling. Their comparative performance across diverse CA architectures and traffic scenarios forms the core of our evaluation.

3.1 Dynamic Algorithm Tuning

The core purpose of the dynamic algorithm is to adjust its refresh timing in response to CA stability indicators. Depending on CA behavior, the algorithm issues different prefetch thresholds that cause the web server to refresh either earlier (to improve reliability under congestion) or later (to preserve CA resources under normal conditions). We evaluated several variants of the dynamic algorithm, including static, adaptive, and aggressive thresholding strategies.

Static Mode The static implementation is the mode that we chose to focus on in our experiments. That is, when the CA is stable, the web server prefetches at a fixed 1 second threshold. Otherwise, it prefetches at 2 seconds. Stability is determined by whether the CA's mean latency exceeds 300 ms or if variance exceeds $20,000$ ms (indicating a very volatile CA server that one must be cautious about). For intuition of these numbers, noticeable lag experienced by end-users is in excess of 50 ms.

Adaptive Mode For adaptive mode, the time we prefetch is not just prematurely tuned to 1 second or 2 seconds. Instead, the prefetch threshold is adjusted on a continuous sliding window between 1 and 2 seconds, inclusive. This calculation is based on the current CA latency, historical latency values for variance (the history buffer that holds these latency values is set to size 50), a max threshold, and a peak threshold. The max threshold is used for calculating the progression between 1 and 2 seconds. The peak threshold is used to help in the situation where our historical latency value distribution is skewed too high.

(1) The prefetch is set to 2 seconds under one of two conditions. The first is when the CA's most recent broadcasted mean latency is above the 90th percentile from the last 50 mean latencies. To avoid the scenario where the historical buffer has too high of latency values (even those below the 90th percentile) that we would consider to be unstable, we also prefetch at 2 seconds if the CA's recent latency is above the pre-defined peak threshold.

(2) When not at peak, the algorithm computes a progressive threshold by adding three calculations together, capped at 3 seconds: a base threshold, a latency-variance penalty, and a stability penalty. The base threshold divides the mean latency by the max threshold and is quantized to fit in the range of 1 to 2 seconds. The latency variance penalty adds up to 0.5 seconds if there exists high variance. Finally, we penalize 0.3 seconds if the CA push notification came back as not stable. This setup allows the algorithm to gradually increase caution as CA degrades.

Aggressive Mode The aggressive mode is analogous to having harder-breaks in the window between stable and unstable waiting times, prioritizing pre-fetching closer to the 1-second mark (for example, instead of a system at 80% stress be linked to a pre-fetch time of 1.8 on a scale of 1 to 2, where 2 is the unstable pre-fetch time, we aggressively set our pre-fetch to 1.5 seconds before expiration). This mode explicitly computes thresholds between 1 and 2 seconds. It uses the following three tiers:

(1) 1.0s when CA is behaving normally
(2) 1.5s when latency crosses 80% of the max threshold
(3) 2.0s when latency crosses the peak threshold

Parameter Sweeps As an ablation study, we evaluated six configurations (DT1-DT6) that varied both the maximum-latency parameter (250-350ms) and the peak-latency threshold (350–450ms). They are defined in Table 1. These sweeps illuminate how dynamic tuning affects the trade-off between prefetch efficiency and soft-fail rates.

Table 1: Dynamic Threshold Configurations

Config	Mode	Max Latency	Peak Latency
DT1	static	300ms	400ms
DT2	adaptive	300ms	400ms
DT3	adaptive	250ms	350ms
DT4	adaptive	350ms	450ms
DT5	aggressive	300ms	400ms
DT6	aggressive	250ms	350ms

The results of the sweep are shown in Figure 2. The results show that soft-fails, from best to worse, include: DT1, DT4, DT5, DT6, DT3, DT2. The prefetch efficiencies show, from best to worse: DT5, DT6, DT2, DT1, DT3, DT4. Overall, the adaptive configuration performs the worst in terms of soft-failure counts and prefetch efficiency. This is conjectured to be due to the fact that its continuous scaling and 50-latency history window make it react to small fluctuations, delaying prefetch unnecessarily. In other words, this variation's attempt to fine-tune timing introduces noise and instability rather than improving reliability. This is further supported by the fact that as we increase the max and peak latency, the adaptive soft-fail count improves. Static, on the other hand, yields the lowest soft-fail count (with only 2), but aggressive mode yields the best prefetch efficiency at 85.9% and 84.8%. The aggressive configuration's results can be attributed to its more conservative definition of what constitutes an unstable CA. In essence, the aggressive configuration aims to keep the prefetch window as narrow as possible, therefore prioritizing the prefetch efficiency over the soft-fail rate. Since our system is trying to optimize for reduced soft-fails, we decided that use the static dynamic configuration for our following experiments.

4 Experimental Design and Implementation

To evaluate the behavior of different certificate prefetch scheduling algorithms under realistic operating conditions, we construct a four-component distributed simulation that mirrors the architecture of real-world OCSP stapling deployments. Our environment includes: (1) a population of clients generating periodic requests, (2) a single web server responsible for managing and refreshing its OCSP staples, (3) one or more CA servers that issue certificates and report their operational status, and (4) a load balancer that distributes certificate requests across multiple CAs. The full system is implemented in Python using socket-based asynchronous communication.

(1) **Clients** We simulate 1000 clients, each representing an independent browser-like process that periodically requests content from the web server. During each request, a client retrieves the stapled certificate currently cached by the web server and records whether it was fresh or stale. Clients do not interact with the CA directly, consistent with the intended design of OCSP stapling.
(2) **Web Server** The web server acts as the central decision-maker for OCSP prefetch scheduling. It maintains a local cache of stapled certificates (one for each hosted website) and is responsible for triggering renewals before expiration. The server runs one of the three scheduling algorithms (baseline, greedy, dynamic) via a background PrefetchScheduler daemon. This scheduler periodically inspects certificate expiration times, ingests CA push notifications, and determines the appropriate moment to request a fresh OCSP response. All certificate requests are routed through the CA or load balancer. No shortcuts or privileged knowledge are introduced.
(3) **CA Servers** We implement one or two CA servers depending on the experiment. Each CA issues certificates, renews them upon request, and periodically sends push notifications to the web server containing mean response latency, latency variance, traffic volume, and a computed stability flag. These metrics are derived from the CA's rolling observations of incoming request loads and simulated network delays.
(4) **Load Balancer** In multi-CA experiments, a load balancer serves as the front-end entry point for all certificate requests from the web server. The balancer selects a CA based on current stress indicators, distributing requests across CAs to avoid hotspots. The load balancer is particularly important for testing whether redundancy alone mitigates stale-staple risk or whether intelligent scheduling still provides measurable benefits.

The load balancer therefore needs to establish communication channels both with the web servers and the CA servers. The web server needs to go through the load balancer first to be routed to a CA server to begin with, and that routing

decision is informed by the same push notifications that the CA server also sends to the web server to inform it of when to prefetch. For the routing algorithm itself, the candidate CA is chosen by examining the traffic volume and mean latency of the CAs and calculating

$$\text{CA Score} = 0.5 \times (\text{traffic volume}) + 0.5 \times (\text{mean latency})$$

The CA with the lower CA score will be routed to. In the event that no push notifications have not been received yet when the web server makes its first request, the load balancer falls back to picking randomly between the provided CAs. Additionally, similar to the previous class implementations, the load balancer makes use of threads to listen for the web server and CA connections (from the push notifications). Because the load balancer needs to handle all web server requests in addition to CA notifications, we decided to implement a thread pool (with 10 worker threads in accordance with our local testing environments) in order to mitigate the overhead impact of spawning a new thread for each web server / CA server response.

4.1 Experiment Variables

We define in Table 2 several sets of experiment variables, which are used to fine-tune the web-server and CA-server characteristics. In essence, variables in each of the four categories move from producing a less-constraint system to a more constrained system.

Table 2: Experimental Variable Definitions

Variable	Category	Values
W1	Web Server	1 certificate; expiry window (0–7 days)
W2	Web Server	10 certificates; each expiry window (0–7 days)
W3	Web Server	50 certificates; each expiry window (0–7 days)
WT1	Web Traffic	Uniform traffic, no spikes
WT2	Web Traffic	Oscillating traffic with periodic spikes
WT3	Web Traffic	Sigmoid-shaped spike with sustained high load
C2	CA Server	Two CA servers with load balancer
C1	CA Server	Single CA; no load balancing
CT1	CA Traffic	Uniform, no spikes
CT2	CA Traffic	Oscillating spikes
CT3	CA Traffic	Sigmoid spike, sustained high load

We define in Table 3 variants of prefetching scheduling algorithms that the singleton web server will utilize.

Table 3: Table of Algorithms

Variable	Description
A1	Baseline
A2	Greedy
A3	Dynamic

Finally, Table 4 defines the experiments we will run, using an unique combination of the experiment variables, along with justification. The experiment tuples are created from a selection of the experiment variables, and the tuple is tested on all three algorithms (A1, A2, and A3).

Table 4: Table of Experiments

No.	Experiment Tuple	Reason
1	(W3, WT3, C1, CT3)	This setup is the most resource-constrained, with both the web server and a single CA sustaining a consistent high load of traffic along with the largest amount of certificates. These results will help isolate how each of the algorithms perform with respect to soft-fails and prefetch efficiency.
2	(W3, WT3, C2, CT3)	This experiment is the same as experiment #1 with the exception that it now uses two CA servers and a load balancer. The intent of the load balancer is to reduce the pressure handled by the individual CA servers, thus hoping to improve the latencies and by extension and soft-fails and prefetch efficiencies.
3	(W1, WT1, C1, CT1)	This setup is one of the least constrained settings, with both the web server and CA experiencing uniform traffic and there only being 1 certificate the web server is querying for. This experiment in comparison with experiment #1 will help us evaluate how robust the dynamic algorithm is. That is, under significantly different traffic conditions, does the dynamic algorithm continue to outperform the baseline? Is the dynamic algorithm suitable for just heavy traffic scenarios, or is it also advantageous to use with steady, predictable traffic?

To justify the realism in experimenting with these tuples, we display the following subset of examples below:

(1) (W1, WT1, C1, CT1): a small one-page business site (one cert) exists on a shared hosting provider with steady, low traffic. The site relies on a single CA endpoint for stapling and there are no traffic spikes. Many legacy, low-traffic websites (brochure sites, internal dashboards) see predictable traffic and use a single certificate/CA.
(2) (W1, WT2, C1, CT1): a singular news blog (one cert) who sees regular daily bursts (diurnal pattern of morning and evening) and occasional viral spikes.
(3) (W2, WT2, C2, CT2): a mid-sized company hosts 10 subdomains (multi-certs, such as for apps, staging, and api handling). Due to the company's localized time zone, traffic

oscillates around business hours. The company gets its certifications from two CA endpoints behind a load balancer. Because these CAs are shared services handling global demand requests from other other companies around the world, this results in the CAs traffic to oscillate over the span of a few hours.

(4) (W3, WT3, C1, CT3): as a final example, a large e-commerce platform hosts 50 certificates for a variety of services (staging, development, production sites, subdomains, api handling). Web traffic experiences a major extended spike due to sale offerings, such as Black Friday weekend: while requests start low, they rapidly increase and then remain at a sustained high level for several days. The company relies on a single CA server, which unfortunately is also under high load from global demand at this time of year, creating potential latency and soft-fail risks for OCSP stapling.

4.2 Evaluation Criteria

To compare the performance of the four prefetch scheduling algorithms, we evaluate each experiment using two primary metrics: prefetch efficiency and soft-fail rate. Together, these metrics capture the trade-off between minimizing stale certificate responses and avoiding unnecessary load on the CA infrastructure.

(1) **Prefetch Efficiency**
Prefetch efficiency measures how effectively the web server schedules certificate refreshes relative to the true expiration timeline. For each certificate, we count the number of prefetches that occur while the existing certificate is still sufficiently valid (prefetches that were not strictly necessary to avoid a stale response). Formally:

$$\text{Prefetch Efficiency} = 1 - \frac{\text{Number of Unnecessary Prefetches}}{\text{Total Prefetches}}$$

An unnecessary prefetch is one triggered far earlier than needed, or one repeated due to CA instability or algorithmic aggressiveness (as often observed with the greedy algorithm). Higher values correspond to more efficient, well-timed refreshes that avoid overloading the CA. In particular, we classify a prefetch as being unnecessary by having the web server fetch for a certificate at a fixed 1 second mark before the expiration time. If the CA is able to respond to that request on time, then prefetches that were triggered earlier are counted as unnecessary (because they would have theoretically succeeded at the 1 second mark as well).

(2) **Soft-Fail Rate**
Soft-fail rate quantifies how often clients receive a stale OCSP staple a certificate whose expiration time has elapsed before the web server successfully refreshed it. A soft-fail occurs when time_left ≤ 0 at the moment the client requests the certificate. We compute the soft-fail rate as:

$$\frac{\text{Number of Client Requests Receiving a Stale Certificate}}{\text{Total Client Requests}}$$

Lower soft-fail rates indicate better reliability and a stronger security posture, as stale OCSP responses are exactly what real-world browsers must soft-fail on, accepting them out of necessity but weakening revocation guarantees.

Together, these metrics allow us to distinguish between algorithms that minimize stale responses and those that balance freshness against CA load. This perspective is essential for understanding how certificate refresh strategies behave under variable traffic, expiration distributions, and CA architectures.

5 Results and Evaluation

All results per experiment are illustrated in Figures plotted at the end of this document. Each result was run multiple times to ensure good analysis. The code and samples of runs from the results can be viewed on the public repository by clicking here.

Experiment 1 Figure 3 looks at the most constrained-styled system to see how our algorithm compares. Over the experiment duration, the soft-fail counts in order of least-to-most are as follows: dynamic (4), baseline (9), then greedy (9). In addition, the prefetch efficiencies in order of most-to-least effective are as follows: dynamic (0.62), greedy (0.25), and baseline (0.06). **Therefore, the dynamic algorithm appears to be best-adapted for heavily-constrained systems.**

These results are in line with what we hypothesized and reinforce the intended purpose of the dynamic algorithm. That is, the system is able to react to instability signals and shift the refresh window to provide a meaningful advantage. This is in contrast to the baseline algorithm which generates the same amount of traffic regardless of the CA's stability. With our current thresholds, the dynamic algorithm falls back to the baseline algorithm when the CA is congested but defers the prefetching if the CA is stable. While this does not allow the dynamic algorithm to avoid congestion once it has already occurred relative to the baseline algorithm, it reduces the likelihood that the system becomes congested in the first place by staggering the refreshes and avoiding premature traffic.

Experiment 2 Figure 4 looks to see how a load balancer with two CAs performs, still under a relatively constrained system. The prefetch efficiencies in order of most-to-least efficient are as follows: baseline (0.91), dynamic (0.7), and greedy (0.5). The soft-fail counts in order of least-to-most are dynamic (6), baseline (16), and greedy (27). Greedy's poor performance can be attributed to its rather aggressive nature and additionally highlights a current weakness of the system infrastructure. Since the greedy algorithm forces the web-server to prefetch every second, it needs to send far more requests to the CA server. This issue is amplified when we introduce a load balancer because the load balancer now serves as a bottleneck point that can be further slowed down due to queuing delays from the thread pool. The results still show that the **dynamic algorithm is still better suited for minimizing the number of soft-fails, even in complex systems that incorporate components like load balancers**.

One thing to note is that the baseline algorithm appears to outperform the dynamic in terms of prefetch efficiency. Initially, this may indicate a potential tradeoff between soft-fail and prefetch

efficiency. However, since the soft-fail and prefetch efficiency trade-off between baseline and dynamic isn't present in experiment #1, we hypothesize that it is a symptom of the CAs experiencing less pressure. Recall that prefetch efficiency is measured by issuing a dummy prefetch request 1 second before a certificate's expiration time and checking whether the CA is able to respond in time. By this definition, an earlier prefetch that would have also succeeded at the 1 second mark is counted as unnecessary. Introducing a load balancer fundamentally changes how often the CA appears stable. Even when the dynamic algorithm detects instability and triggers a prefetch at 2 seconds, the load balancer can help the system recover more quickly by distributing requests across the two CAs. The CAs are subsequently back to being stable by the 1 second mark more frequently. This means that many of the dynamic algorithm's earlier prefetches would have still succeeded had the server waited. Thus with this particular metric scheme, those prefetches are counted as unnecessary and contribute to the lower prefetch efficiency as illustrated in Figure 3.

In comparison to Experiment 1 (the setting C1, which has no load balancer), the algorithms under perform in terms of soft-fail counts when compared with their corresponding experiment pairing. This is contrary to what we hypothesized because the load balancer was supposed to distribute the traffic amongst the CAs more evenly, ensuring that the CAs don't experience as much pressure and as early as they do for C1. A clue for why this may be case comes from the fact that we experience soft-failures at an *earlier* point, and an earlier onset of soft-failures would enable more accumulation as the experiment goes on. This is illustrated by the baseline and dynamic algorithms. For the dynamic algorithm, in this experiment (C2), the first soft-fail is seen before 10 seconds, whereas in the first experiment (C1), the first soft-fail was seen after 50 seconds. Note that for the dynamic model, however, that in the C1 case, once the first soft-fail is experienced, the number of soft-fails rapidly increases compared to the C2 setting, where the progression of soft-fails is more staggered and stable for certain intervals. This suggests two conclusions: (1) the system nature of the load balancer does introduce overhead that causes earlier onset pressure, and (2) under this system and a dynamic model, we're able to mitigate the rapid accumulation of soft-failures seen in C1.

Experiment 3 Figure 4 looks to see how the algorithms work along the least-constrained system. Generally, all three algorithms appear to be the same in terms of prefetch efficiency and soft-fail rate: by the end of the experiment, the results as tuples (soft-fail, prefetch efficiency) is as follows: baseline (10, 0.685), greedy (9, 0.714), and dynamic (10, 0.83). What we see is that **the dynamic prefetch scheduler algorithm does not perform any worse than the others when the system does not require us to have a necessarily complex algorithm, which helps provide a good buffer in its usage**. It appears then that the dynamic algorithm is more advantageous in high-constrained systems. These results align with our expectations because with little traffic, the CA will remain stable for most of the time and thus produce little variation in the performances of the algorithms.

Why are the prefetch efficiencies relatively high, especially when compared to experiments #1 and #2? In a less constrained system, a CA should be able to handle requests that come closer to the expiration time without an issue, indicating better stability. Yet, the algorithms will still fetch before the metric we use to determine the CA's stability, particularly for baseline, which always prefetches ahead of when this metric is meant to run. This would then predict a *lower* pre-fetch efficiency. One possible explanation for this is that even though the CA is handling a lighter load and should be stable most of the time, it could still experience near-threshold jitter towards the expiration boundary that would cause 1 second pre-fetches to fail.

6 Limitations of Study and Future Work

The following sections detail potential continuations of this paper and our study's current limitations.

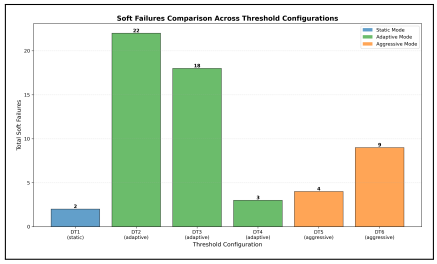
Hyper-parameterization While we provide justification for the various thresholds used in our prefetch scheduling algorithms, certificate revocation timeouts, and CA stability metrics, real-world behavior is often more complex and less predictable. The chosen parameters may not capture all variability observed in live deployments, and so future experimentation with such parameters would be invaluable.

We would additionally like the further explore the adaptive dynamic algorithm. Although it performed poorly relative to the static and aggressive methods, this can likely be attributed to weaknesses in our current configuration rather than a fundamental weakness of the approach. For instance, perhaps the historical window of the adaptive approach should not only consider past mean latencies but also past variance latencies. Its decision thresholds were hand-tuned, so we believe it would be worth exploring a sophisticated tuning approach that would instead rely on learning-based thresholds or even feedback driven methods that adjust not only to the CA traffic, but also to the performance (resulting soft-fails) of previously set thresholds.

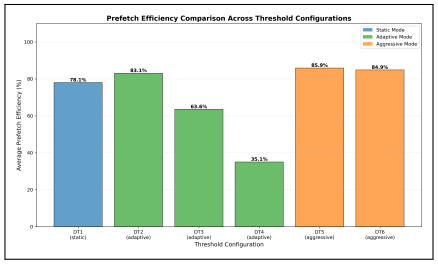
Multiple Web-Servers In our current setup, we represent traffic coming into the CA from other web-servers by using oscillating or sigmoid-styled functions. A more realistic evaluation would involve aggregating multiple instantiated web-server functionalities to impact the traffic of the CA. This may provide deeper insights into how such interactions influence prefetching performance.

Load Balancer Implementation Our current load balancer implementation doesn't fully replicate real-world CA redundancy. Since we implemented our CA servers as Python classes, the C2 experiments currently mimic redundancy across the CAs by using a global dictionary that is accessed and written to by both of the CA objects. This setup prevents us from observing behaviors that would normally arise in distributed CA infrastructures such

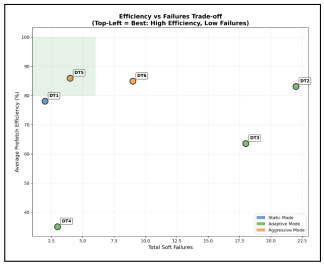
Figure 2: Tuning Results



(a) DT1-DT6 Soft-Failures

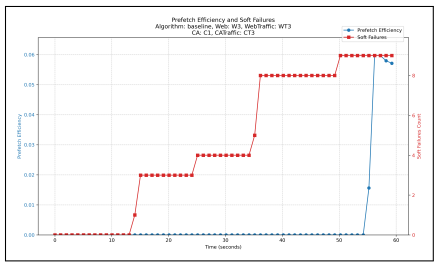


(b) DT1-DT6 Prefetch Efficiency

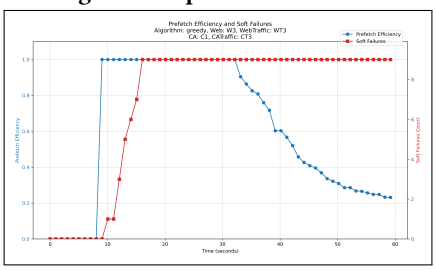


(c) DT1-DT6 Ratio Comparison

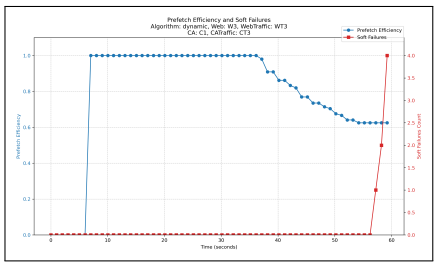
Figure 3: Experiment 1 Results



(a) Baseline

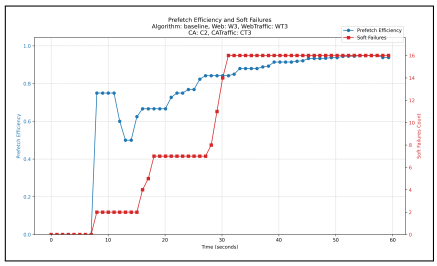


(b) Greedy

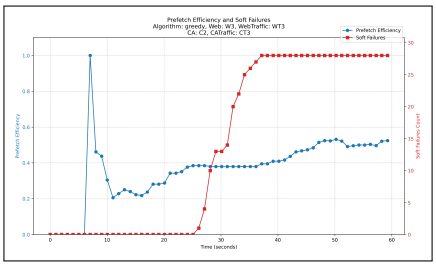


(c) Dynamic

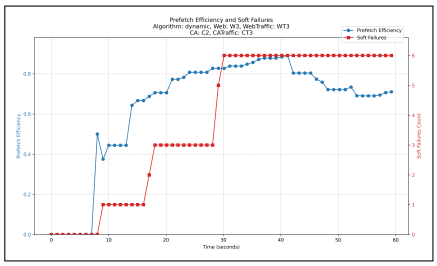
Figure 4: Experiment 2 Results



(a) Baseline

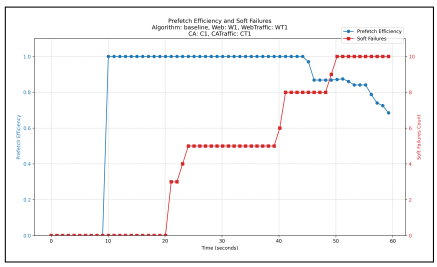


(b) Greedy

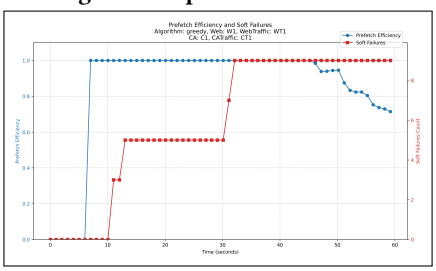


(c) Dynamic

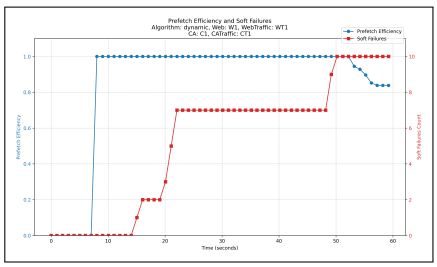
Figure 5: Experiment 3 Results



(a) Baseline



(b) Greedy



(c) Dynamic

as replica divergence and most notably delayed propagation of updated certificates that would only add latency to the system. Future steps could include implementing distributed CA servers such that they all maintain their own internal copies of certificates and then observing how various replica models like master-slave impact the soft-failure and prefetch efficiency rates. Additionally, since the current CA score formula dictates how the traffic is distributed entirely, it would also be worth experimenting with alternative scheduling algorithms to see which one fares the best under varied network conditions. Similar to a caveat with our dynamic algorithm tuning approach, the CA score's weights were hand-tuned. Therefore, it would be worth exploring how a learning-based and feedback driven approach for determining the CA score would cause the system to perform.

7 Conclusion

In conclusion, we proposed a dynamic, CA-aware prefetching algorithm that incorporates CA-to-server push notifications and adaptive refresh timing based on CA performance and web traffic conditions. We experimented with this dynamic algorithm under a simulated distributed environment that included clients, web-servers, CA servers, and load balancers. We found that under heavily-constrained systems, the dynamic algorithm overall reduces the soft-fail rates while balancing the prefetch load more effectively than other scheduling algorithms. As the system design became more stable, the dynamic algorithm helped to smooth out earlier spikes in soft-fails and was always either generally at-par or better in soft-fails than the other algorithms. Our results therefore offer a viable alternative to strict must-staple policies, with promising areas for future improvement.

Acknowledgments

We would like to express our gratitude to Professor James Mickens for his engaging lectures, thoughtful feedback, and for inspiring our interest in systems security throughout the course.

References

[1] SSL Insights [n. d.]. *SSL/TLS Certificates Statistics 2025.* SSL Insights.
[2] David Cerenius, Martin Kaller, Carl Bruhner, and Niklas Arlitt, Martinand Carlsson. [n. d.]. *Trust issue(r)s: Certificate revocation and replacement practices in the Wild.* SpringerLink.
[3] Laurent Chuat, AbdelRahman Abdou, Ralf Sasse, Christoph Sprenger, David Basin, and Adrian Perrig. 2020. SoK: Delegation and Revocation, the Missing Links in the Web's Chain of Trust. Department of Computer Science, ETH Zurich. https://netsec.ethz.ch/publications/papers/sok-delegation-revocation.pdf
[4] T. Chung, J. Lok, B. Chandrasekaran, D. Choffnes, D. Levin, B. Maggs, A. Mislove, J. Rula, N. Sullivan, and C. Wilson. 2018. Is the web ready for OCSP must-staple?. In *Internet Measurement Conference.* University of Maryland Department of Computer Science. https://www.cs.umd.edu/~dml/papers/ocsp_imc18.pdf
[5] Steven Davidson. [n. d.]. *TLS Certificate Lifetimes Will Officially Reduce to 47 Days.* Certificate Management.
[6] Robert Duncan. [n. d.]. *Certificate revocation and the performance of OCSP.* Netcraft.
[7] Samantha Frank. [n. d.]. *Scaling Rate Limits to Prepare for a Billion Active Certificates.* Let's Encrypt.
[8] Nick Sullivan. [n. d.]. *High-reliability OCSP stapling and why it matters.* Cloudflare, Inc.