# Characterizing JSON Traffic Patterns on a CDN

Santiago Vargas
Stony Brook University
savargas@cs.stonybrook.edu

Utkarsh Goel
Akamai Technologies
ugoel@akamai.com

Moritz Steiner
Akamai Technologies
mosteine@akamai.com

Aruna Balasubramanian
Stony Brook University
arunab@cs.stonybrook.edu

## Abstract

Content delivery networks serve a major fraction of the Internet traffic, and their geographically deployed infrastructure makes them a good vantage point to observe traffic access patterns. We perform a large-scale investigation to characterize Web traffic patterns observed from a major CDN infrastructure. Specifically, we discover that responses with `application/json` content-type form a growing majority of all HTTP requests. As a result, we seek to understand what types of devices and applications are requesting JSON objects and explore opportunities to optimize CDN delivery of JSON traffic. Our study shows that mobile applications account for at least 52% of JSON traffic on the CDN and embedded devices account for another 12% of all JSON traffic. We also find that more than 55% of JSON traffic on the CDN is uncacheable, showing that a large portion of JSON traffic on the CDN is dynamic. By further looking at patterns of periodicity in requests, we find that 6.3% of JSON traffic is periodically requested and reflects the use of (partially) autonomous software systems, IoT devices, and other kinds of machine-to-machine communication. Finally, we explore dependencies in JSON traffic through the lens of ngram models and find that these models can capture patterns between subsequent requests. We can potentially leverage this to prefetch requests, improving the cache hit ratio.

## CCS Concepts

• **Networks** → *Network measurement*.

## Keywords

Content Delivery Networks (CDNs), JSON, Web

**ACM Reference Format:**
Santiago Vargas, Utkarsh Goel, Moritz Steiner, and Aruna Balasubramanian. 2019. Characterizing JSON Traffic Patterns on a CDN. In *Internet Measurement Conference (IMC '19), October 21–23, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3355369.3355594


*IMC '19, October 21–23, 2019, Amsterdam, Netherlands*

ACM ISBN 978-1-4503-6948-0/19/10…$15.00
https://doi.org/10.1145/3355369.3355594

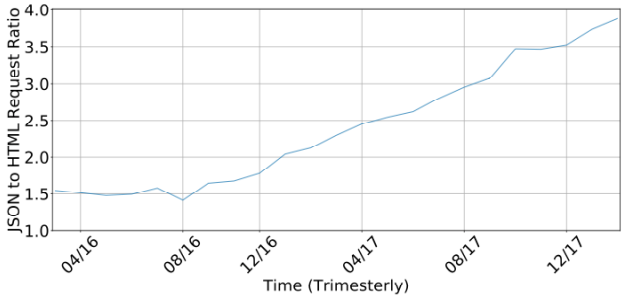


**Figure 1: Ratio of JSON to HTML requests on the CDN.**

## 1 Introduction

Content delivery networks (CDNs) serve a large fraction of Internet traffic, even more than 50% of all Web traffic for some carriers [16, 23]. CDNs are also a core part of Internet infrastructure used to optimize, secure, and reliably deliver content. As a result, CDNs are a good vantage point to observe large scale Internet patterns, which are constantly changing [28].

We study the traffic patterns in a large CDN and observe one such changing pattern: that of the growth of the JSON content type. JSON, which stands for JavaScript Object Notation, is a format used for transferring data in key-value pairs [5]. We find that JSON has outgrown HTML, CSS, and JS content in the last 3 years, and is the leading content type on the CDN. Figure 1 shows the growth trend of JSON requests to HTML requests received by the CDN since 2016 using counts of the total number of JSON and HTML requests recorded by all CDN edge servers. At the end of the observation period, JSON is requested more than HTML content by over **4**×.

However, little is known about how JSON is used on the Web. HTML, which has been one of most popular content types for Web traffic, is well studied. As a result, there are several optimization techniques designed to optimally deliver HTML content. These optimizations range from rewriting HTML [21], prefetching [27], and pushing sub-resources [13]. However, the same is not true for JSON. Our goal in this paper is to shed light on what JSON traffic looks like from the vantage point of a CDN. Specifically, we seek to answers to the following questions:

- What applications and devices are consuming JSON traffic?
- What are the types of JSON requests and responses and what are their properties?
- What are the common patterns of JSON traffic? Can these patterns be exploited for optimization?

To answer the above questions, we analyze the logs of a total of 35 million JSON requests collected from the edge servers of Akamai's network over a 24 hour period. We start by creating a taxonomy of JSON traffic properties based on the data we collect. Specifically, we divide the properties of JSON traffic into traffic source, request type, and response type and analyze each component individually.

In our data, 88% of JSON traffic is non-browser traffic, and only 12% is requested by browsers. At least 52% of JSON traffic is also from native mobile applications. This result is particularly important because browser traffic is guided by an HTML manifest file. This provides optimization opportunities including prefetching and server push. However, non-browser traffic from mobile apps are less standardized and harder to optimize. Further, we find that at least 12% of JSON requests come from embedded devices, including game consoles and smart watches. These segments of devices are often bottlenecked in terms of network performance.

Our observations in terms of request/response is that 55% of JSON requests are not cacheable, Thus, the slower mobile and embedded devices are largely not benefiting from caching optimization. In fact, we find that 50% of domains do not use CDN caching at all. Instead, these domains use other security and performance products for their JSON traffic. Cacheability also depends on domain industry category. Media, News, and Sports serve highly static, unchanging content. Conversely, Financial, Streaming, and Gaming domains serve JSON that is personalized or meant for one-time use.

Finally, our examination of the JSON traffic in mobile applications shows two common traffic patterns. First, there is a non-trivial amount of periodic JSON traffic from mobile applications. Second, we find that, once a JSON object is requested, the next request can be predicted with some accuracy. JSON requests from mobile apps in fact serve as manifest files that contain references to further JSON objects. We explore these two patterns in this paper to present directions for JSON traffic optimization.

We look at the most requested objects and find that 6.3% of JSON requests are periodic. Examples include pulling latest messages in a messaging system, periodically updating scores for online gaming, and telemetry reporting. For more than 20% of objects that are periodically requested, 50% of the clients that request these objects do so with matching time signals. This periodicity suggests that the requests are coming from autonomous software systems, IoT devices, or other kind of machine-to-machine communication. Since these requests are not human-triggered, one possible optimization is to de-prioritize these requests.

Second, we find that in many cases, a JSON request can predict a subsequent JSON request with about 70% accuracy. Prediction of these objects can be used to prefetch future requests in the case of cache misses and unchangeable content. Further, up to 87% accuracy is possible when clustering similar requests by URL. This shows that clients share general patterns across requested objects. Apart from performance, prediction of clustered objects can also be used for anomaly detection of unusual requests.

While we characterize JSON traffic and look at two patterns inside of the traffic, we have just scratched the surface of analyzing JSON traffic. The increase in JSON traffic is an important trend with implications on the Internet. Since JSON traffic has grown quickly in the last years, it is important to analyze this trend and understand this new segment of traffic.

| **JSON Manifest Traffic Pattern** |
|---|
| 1. Request: **GET** → *news_example.com/stories* |
| Response: ← "application/json"<br>[{"article_id": 1234,<br>"article_title":"Lorem Ipsum",<br>"image_url":"news_example.com/image1234.jpg"<br>}, ...] |
| 2. Request: **GET** → *news_example.com/article/1234* |
| Response: ← "application/json"<br>{"video": "news_example.com/video1234.mp4",<br>"article":"Lorem ipsum dolor...",<br>"images":["image1_url", "image2_url"]<br>} |

**Table 1: Example of a mobile news app using JSON to request 1) a summary of stories and 2) content for a specific article.**

## 2 Background

In this section, we first describe the JSON format [5] and then describe why and how applications request JSON objects.

### 2.1 What is JSON?

JSON, which stands for JavaScript Object Notation, is a format used for transferring data in key-value pairs. JSON is more lightweight than formats like XML and HTML, which require opening and closing tags. But, JSON retains the same benefits as XML and HTML of being text-based and hierarchical unlike plain-text and binary formats. Because JSON is structured, standardized, and lightweight, this makes it a portable format that can be used on many platforms. Further, JSON can be parsed and programmed against in any environment that runs JavaScript [1].

### 2.2 Why use JSON?

Traditionally with web and media content, both the view (layout) and data are not cleanly separated. For example, server-side rendered HTML contains information about both the layout and the data inside the web page; not all layout information is contained in the style sheet (CSS). However, with the advent of web and mobile applications, many developers have taken to separate the client view, application logic, and application data, such as with progressive web applications [9] on browsers. Developers can then control when and how often to update both application views and data separately. Since each individual component of an application need not be updated, bandwidth overhead is reduced by caching unchanged views or unchanged data. Thus, these applications generally cache static application layouts and logic at the client. Meanwhile, data is transported between client and server in a specified data format, such as JSON.

### 2.3 JSON Traffic Examples

Based on popular applications that use JSON in the CDN, in Table 1 we illustrate two examples of how JSON is used. 1) Applications request a JSON manifest object that contains direct (URL) or indirect (object ID) references to other objects. In the case of a popular news application, we observe that the application first retrieves a JSON manifest containing references to text and image contents for many

| Dataset | # of Logs | Duration | # of Domains |
|---|---|---|---|
| *Short-term* | 25 million | 10 mins | ~5K |
| *Long-term* | 10 million | 24 hrs | ~170 |

**Table 2: Summary of our datasets.**

articles. Afterwards, subsequent content is retrieved, like full text, images, and videos for select news articles. 2) Applications periodically send requests to a webservice for telemetry purposes. For example, some applications send requests in a polling-like behavior for tracking or advertising.

## 3 Methodology

In this section, we describe the platform used to collect network request data and outline the significance of individual data fields.

### 3.1 Data Collection

This study uses server logs gathered from Akamai, a leading CDN network. The Akamai network consists of around 240k distributed servers in about 140 countries and 1600 networks worldwide [3]. Additionally, the Akamai CDN serves around 3 trillion HTTP requests daily [25]. This network has a large market share of Fortune 500 companies and top Alexa sites.

Each time a client makes an HTTP request to an Akamai edge server, a request log is generated at the server. We collect logs from CDN edge servers. The request information we collect from each log includes the time of the request, object caching information, a client IP address that is hashed for anonymity, and select HTTP request and response header information including user-agent, mime type, and object URL.

Since it would be infeasible to collect request data for the entire Akamai network over an extended period of time, we collect two datasets, a shorter, wider dataset and a narrower, but longer-term dataset. Note, since traffic patterns are constantly changing, our datasets may be influenced by the capture lengths and locations. Table 2 summarizes the two datasets we collect. The short-term dataset is collected over all machines in the entire CDN network to have a large coverage of diverse network traffic. We use this dataset for generalized JSON characterization in §4. The long-term dataset is collected from all machines in three CDN vantage points in Seattle, WA to record requests to objects from a subset of domains for a longer period of time. Since this dataset covers long periods of time, we use this dataset to analyze patterns in §5.

### 3.2 JSON Traffic Taxonomy

As a first step, we define a taxonomy for JSON traffic to more easily study the traffic. We categorize the traffic based on the traffic source, request type, and response type. Figure 2 shows this taxonomy. Below we define each property of the taxonomy and explain how CDN request logs data captures the properties.

**Identifying Content Type:** First, we use the HTTP mime type header field to identify the content-type of traffic. Content-type values are standardized by the IANA and follow a specific format [7]. Applications, such as browsers, rely upon the content-type of a request to determine what type of content is being downloaded

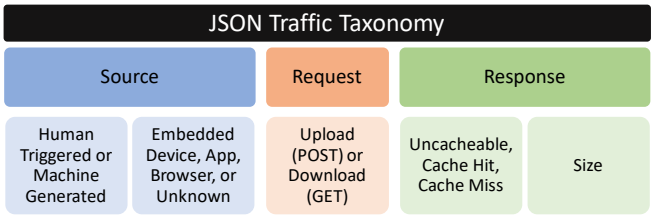


**Figure 2: JSON traffic taxonomy.**

[8]. To filter for JSON traffic, we only include HTTP requests that contain "application/json" in the HTTP mime type header.

**Traffic Source:** Traffic source describes the initiator of a JSON request. Two properties of traffic source are 1) human-triggered vs. machine-generated traffic and 2) device type.

Human-triggered traffic is traffic produced as a direct interaction between a human with a system. For example, in the case of a human interaction that opens a news app on a smartphone and loads an article which triggers the JSON traffic in Table 1, this traffic is considered human-triggered JSON traffic. All other JSON traffic is considered machine-generated traffic, such as traffic that is automatically generated from a script. We use timing information to analyze this traffic in §5.1.

Another property of the JSON traffic is device type and application type. For device type, we consider the following categories: mobiles, desktops/laptops, and embedded devices. Embedded devices are non-mobile, non-desktop devices, such as game consoles, IoTs, smart TVs, etc. We use the user-agent HTTP header field to identify devices and applications that generate traffic similar to previous work [18, 33]. To identify device type, we group by system identifiers in the user-agent field, such as "Android", "iPhone", "Windows", etc. Note, since the user-agent field is not standardized across many platforms and other applications, there may be false positives and negatives as well as missing values when identifying device-type using the user-agent field. To reduce misclassification, we use Akamai's EDC database to further extract device characteristics [2]. We also label the traffic source as *Unknown* when a user-agent is missing or is unidentifiable. To separate between browser and non-browser traffic, we use a database of browser user agents [11] since browsers use well-formed user-agent strings.

**Request Type:** In the taxonomy, we outline two types of JSON requests, uploads and downloads. Upload requests are used to send data from a device to a server, and downloads retrieve data from a server. To distinguish between upload and download requests, we use the HTTP method header of a request. By convention the GET method, as the name suggests, does not send data and is used to download content. Likewise, the POST method uploads data and also receives JSON object responses. Though clients can deviate in behavior, this study assumes conventional client behavior of GET and POST methods for uploads and downloads respectively as specified by the IETF [15].

**Response Type:** There are 2 aspects of the response type: size and cacheability. Response size relates to the number of bytes served per each response and directly affects the CDN's costs of serving JSON requests. For cacheability, CDN cache logs give insights on the type of traffic that is being served. CDN customers decide whether a

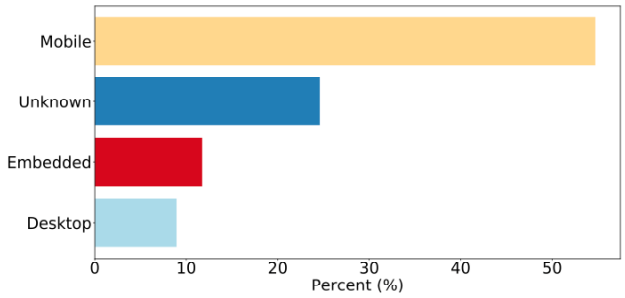


**Figure 3: Categorization by device type.**

response is cacheable and provide this information to Akamai's edge cache system for use in serving responses from cache or customer servers. If JSON traffic is not cacheable, then it is dynamic content and/or personalized. If traffic is either not cacheable or a miss on the CDN, then the response is provided by the CDN customer's own infrastructure.

## 4 Characterizing JSON Traffic

We analyze and categorize the JSON traffic according to the taxonomy in Figure 2.

**Traffic Source:** Figure 3 shows the breakdown of JSON requests in terms of traffic source. Mobile smartphones and embedded devices account for at least 55% and 12% of all JSON traffic respectively. Within the embedded devices, we observe smart watches, game console, and smart TVs. Finally, 24% of traffic is *Unknown*, meaning that the traffic either does not contain a user agent (most of the traffic) or the user agent cannot be linked to a platform. We also find that the distribution of user agent strings is as follows: 73% of UA strings are Mobile, 17% are Embedded, 3% are Desktop, and the remaining 7% are Unknown.

Interestingly, 88% of the JSON traffic is *non-browser traffic* (not shown in figure). If we specifically look at mobile traffic, mobile browser-based traffic is 2.5% of all requests. No browser traffic is detected on embedded devices. The takeaway is that a large portion of the traffic includes native applications on mobiles and embedded devices. Browser traffic has a well known pattern that is derived from the HTML template; the result is they can be optimized using prioritized push strategies [13] or prefetching [27]. However, mobile applications are independent entities that tends to have varied traffic patterns.

**Request Type:** In the taxonomy, we identify two types of JSON requests, uploads and downloads based on the GET and POST HTTP methods. We find that 84% of requests are GET requests meaning that the majority of JSON traffic is download traffic. 96% of the remaining requests are POST traffic, which uploads data values to the server. We further explore the impact of uploads vs. downloads as they relate to cacheability in the next section.

**Response Type:** The CDN allows customers to configure cacheability of each individual request. If a request is for an uncacheable object, the request must propagate from the edge server through the CDN to origin content servers in order to obtain a response. Then, the response object is returned to the original edge server to be returned to the requesting client.

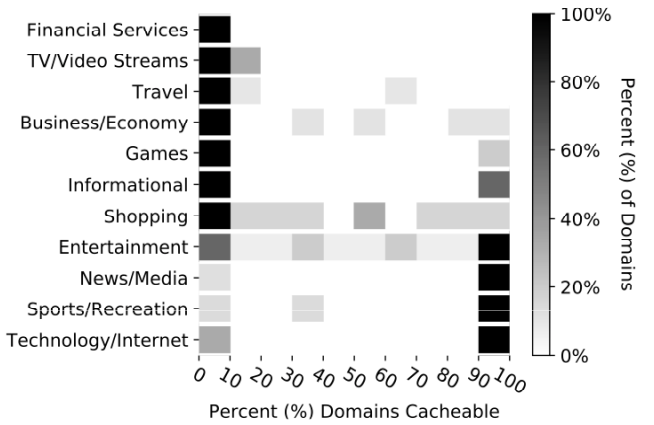


**Figure 4: Heatmap of domain cacheability by category.**

Nearly 55% of all JSON traffic is not cacheable. This means that more than over half of the JSON requests, the most popular CDN content-type, are tunneled through the CDN to origin servers. Another interesting aspect is that the average size of JSON responses has decreased by around 28% since 2016 (not shown in figure). JSON objects are also 24% and 87% smaller than HTML responses at the median and 75th percentiles respectively. Reduced response sizes increase the CPU cost-per-byte of serving JSON traffic, since a large chunk of the total request cost (CPU, network, IO, etc...) is tied to CPU request processing, which must be taken into account by network operators when provisioning the network.

We further categorize the domains in our dataset according to the industry associated with the domain using a commercial service [10]. Figure 4 shows a heatmap of domain cacheability grouped by the associated domain category for the top 11 domain categories. First, nearly 50% of domains serve content that is never cacheable and another 30% serve content that is always cacheable. Therefore, many CDN customers that serve JSON do not cache on the CDN and instead use Akamai for traffic performance, security, and analytics [4, 6, 22].

Second, our categorization shows that specific segments of industry generally hold similar business patterns. For example, Financial Service, Streaming, and Gaming domains are not cacheable since these services serve one-time use or personalized content. Conversely, the majority of News/Media, Sports, and Entertainment domains are mostly cacheable since their content is highly static.

## 5 JSON Request Patterns

We observe two patterns in JSON requests that have implications on optimizing JSON and discuss them below.

### 5.1 Periodicity

We first study periodicity in the JSON traffic. The difficulty in finding periodicity in network traffic is noise in traffic signals. Because of network and program delays, one cannot rely on cleanly receiving a script period. Instead, we extend existing previous signal processing techniques [29] that only show the significant periods. The key idea is to use a combination of autocorrelation (on the time domain) and fourier transform (on the frequency domain) to extract key periods and randomness to filter noisy periods.

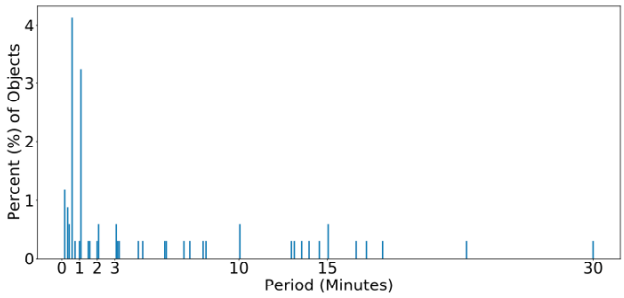


**Figure 5: Histogram of JSON object periods.**

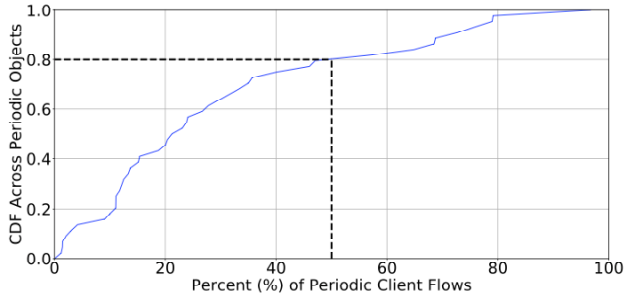


**Figure 6: CDF of the percent of periodic clients across objects.**

Let an *object flow* be the sequence of requests made by all clients to a specific object, identified by a unique URL in the dataset. Let a *client-object flow*, $CO_{flow}$, be a subsequence of *object flow* requests from one client, identified by a user agent and anonymized client IP pair. To obtain significant results, we filter out *client-object flows* with less than 10 requests as well as *object flows* with less than 10 clients, resulting in flows containing the top 25% of objects requested. We extend the signal processing algorithm [29] to our dataset as follows:

(1) Calculate the autocorrelation and fourier transform for each $CO_{flow}$.
(2) Randomly permute $CO_{flow}$ a total of $x$ times and calculate autocorrelation and fourier transform for each permutation, recording the max period and frequency of autocorrelation and fourier transform respectively.
(3) Of all max periods and frequencies, take the $(x-1)^{th}$ largest period and frequency as thresholds for autocorrelation and fourier transform of the original unpermuted $CO_{flow}$.
(4) Using the thresholds to discard insignificant periods and frequencies, line up autocorrelation and fourier transform of $CO_{flow}$ to find the most significant frequency and period as the overall $CO_{flow}$ period.

Note, the above algorithm either returns the most significant period (i.e., largest peak in autocorrelation) or no period for the flow, due to noise thresholding. As a result, we assume a flow only contains one significant period and leave multi-period analysis for future work.

**Choosing Parameters:** We run the above algorithm on all *object* and *client-object* flows to determine a flow period. Note the above algorithm is parameterized by $x$, where larger values of $x$ provide higher accuracy at the expense of computation. We empirically find that values of $x$ greater than 100 do not produce significantly different results in our dataset. Therefore, we use $x = 100$ in our experiments. For autocorrelation and fourier transform, we also set sampling rates of 1 second assuming that accurate detection of periods less than this sampling rate is difficult due to network jitter.

**Results:** If both *object* and *client-object* have periods and these periods match, we label the client flow as periodic with respect to its object. We run the above analysis on our long-term dataset, since we do not want to limit the analysis to short periods, and find that 6.3% of JSON requests are periodic. This translates to a significant share of Akamai's requests given that JSON is the most requested content type. Figure 5 shows that the *object flow* periods detected by the above algorithm are largely on even time intervals. For example, there are spikes are 30$s$, 1$m$, 2$m$, 3$m$, 10$m$, 15$m$, and 30$m$. Figure 6 quantifies how many *clients-object flows* share the same *object flow* period (ie. how many clients all request the same periodicity). Highlighted in the figure, we see that 20% of periodic objects have a majority (>50%) of clients that make periodic requests. Using the metrics from Section 4, we find that periodic traffic is 56.2% uncacheable and 78% upload traffic. Therefore, a large amount of this traffic flows through the CDN to customer infrastructure.

Since these objects are highly requested periodically and the majority of clients share the same period, these objects are likely not requested by humans and are instead programmatic **machine-to-machine traffic**. Though these requests are not definitively machine-generated, they are most likely machine-to-machine requests since it is highly unlikely for a large group of humans to accurately send many request in periodic intervals over sustained periods of time. Understanding this traffic as machine-to-machine allows network operators to isolate and investigate these requested objects in order to apply further management policies and improve user QoE. One possible optimization is for CDN operators to deprioritize machine-to-machine traffic since a human is not waiting for the response. Periodic information can also be used for anomaly detection when an object is requested at a different period than it is intended to be requested.

## 5.2 Request Prediction

The second application pattern shows that JSON requests are amenable to prefetching, since given a request, we can predict the subsequent JSON requests. This usage pattern is similar to other web and mobile app dependencies where previous approaches have used static and dynamic program analysis to discover resource dependencies [14, 20].

Since our JSON traffic is not application or platform specific, we do not use program analysis like prior approaches. Instead, we model the relationship between requests using a backoff ngram model [12]. The ngram model captures transition probabilities from a subsequence of previously requested objects to the next request in the client flow. Though the ngram model does not directly show a dependency relationship between two objects unlike program analysis, it is a data-driven method to empirically show probable object requests. Furthermore, this approach takes into account the popularity of highly requested items, unlike standard program analysis.

**Methodology:** To evaluate this approach, we first split the JSON dataset by unique clients into a testing and training set. As a feature

| | Accuracy | |
|---|---|---|
| **K** | *Clustered URLs* | *Actual URLs* |
| 1 | .65 | .45 |
| 5 | .84 | .64 |
| 10 | .87 | .69 |

**Table 3: NGram model accuracy for URLs with a history of $N = 1$ and varying $K$, where $K$ is a parameter for choosing most probable requests.**

to the ngram, we use request URLs, such that a previous URL transitions to the next URL. Since 84% of JSON requests are GET requests (see §4), they do not carry request bodies and the URL is sufficient to fetch a response. However, request cookies are not considered due to privacy and client fingerprinting concerns. Requests are split into client request flows, ngram transitions are created from individual client requests, and these ngram transitions are used to train the ngram model.

**Evaluation:** We evaluate this method on two ngram models, one with actual, unmodified URLs and one with clustered URLs, using clustering similar to URL argument clustering in [13]. Since 84% of requests are GET requests, unmodified URLs can be used to request these objects directly. However, when URLs contain unique client information, such as client id's or coordinates, clustered URLs can reveal general object dependencies for the application. The ngram models are also tested on individual client request flows.

We examine the change in accuracy when predicting with larger $N$ and $K$ parameters. $N$ denotes how much history to use when predicting where $N = 1$ predicts only using the most recent request (ie. history of 1). $K$ selects the number of requests to predict in probability order.

Table 3 shows that predicting a larger set of requests (larger $K$) greatly improves accuracy even with only immediate history of the previous request, ie. $N = 1$. Using larger $N$ like $N = 5$ only marginally increases accuracy by up to 5%. Overall, using this model we achieve about 70% prediction accuracy for actual URLs. This is a promising result showing that request order is highly probabilistic and hints that more robust machine learning systems may be able to better predict requests. 87% accuracy is also achieved for clustered URL showing that JSON requests exhibit general ordering patterns.

Overall, these results show that a JSON request prediction system can be used by CDNs to perform prefetching for cacheable requests. HTTP Server Push can also be used to preemptively send responses to the client, improving overall response time. Also, the general ordering patterns discovered can be used to better inform current caching or load balancing systems, and to perform anomaly detection (i.e., detect when a highly unlikely object is requested). While our prediction analysis examines request access order, future work can also take into account request interarrival time to better inform prediction systems.

## 6 Related Works

**Web Proxy Optimization:** Given that the CDN can serve as an HTTP proxy, there is a plethora of literature on proxy optimizations for web content. One group of works seek to discover network and computational bottlenecks caused by inter-dependencies in browsing workloads [20, 30]. Researchers have also explored using split-browser architectures, which are modified browsers and cloud proxies work together that selectively offload expensive parts of the web page load to the proxy [27, 31]. Yet, other works improve webpage performance by rewriting pages for efficient execution [21]. Similarly, we explore dependency patterns in JSON traffic through the lens of traffic prediction. In the mobile web space, researchers using program analysis and configuration to proposed local and remote prefetching proxies that improve mobile application quality of experience [14, 17, 35]. By contrast we propose data-driven CDN prefetching to improve JSON traffic delivery performance.

**Network Traffic Characterization:** One line of research looks at characterizing Internet trends using network data logs. Researchers have explored extracting client information from user-agent strings to characterize network traffic [18, 33]. Another vein of work looks at the problem of Mobile Application Identification, mapping network traffic to the applications that generated the traffic [19, 32–34]. We also breakdown JSON traffic by device and application types using user-agent. In [26], researchers use CDN baseline activity for blocks of IP addresses to identify activity anomalies in the traffic for these blocks and determine if there is an internet outage for these IPs. Pujol et al. [24] identify machine-to-machine traffic by checking for valid TCP or HTTP responses from standard web ports. Instead, we find periodicity patterns that identify machine-to-machine JSON traffic.

## 7 Implications & Limitations

In this section, we outline implications and limitations of this study. First, since the number of JSON requests has been growing while the size of the JSON objects has decreased, networking systems should account for changes when serving this type of content and provisioning new infrastructure. Secondly, §5 explains two patterns present in JSON traffic, periodicity and request ordering. We suggest that these patterns can be exploited by network operators to implement management policies for periodic traffic and to build prediction-based systems to improve network performance. While we suggest deprioritization of periodic traffic over, network operators should evaluate the effects of deprioritization on user QoE. Lastly, this study looks at two large-scale datasets collected from Akamai's network, a five minute dataset over Akamai's entire network and a day-long dataset covering one geographic region. Future studies can analyze longer datasets covering more regions in order to explore geographic and temporal differences in JSON traffic patterns. Though this study presents these initial implications, it is important to further analyze the traffic shift to JSON for further implications.

## 8 Conclusion

In this work, we explore the prominence of JSON traffic on a large CDN. We first examine content-types on the CDN and find that JSON content is largely more requested than HTML and there is a growth trend for JSON traffic. Next, we create a taxonomy of JSON traffic and analyze HTTP request logs from CDN edge servers to further explore what types of clients are using JSON content, what types of requests and responses are seen on the CDN, and how we can use JSON patterns to optimize JSON traffic. We find that the majority of JSON traffic is non-standardized traffic that is requested by mobile and embedded devices. We also see that more than half

of all JSON traffic is not cacheable, especially for specific industries. Next, we find that at least 6.3% of JSON traffic is machine-to-machine traffic and CDN operators can deprioritize this traffic as it is not human-triggered and no human is waiting while staring at a screen. Finally, we explore prediction of JSON content and note that since 87% of content can be predicted, prefetching is a viable optimization for JSON traffic.

## References


[1] Ecmascript 5.1 language specification. https://www.ecma-international.org/ecma-262/5.1/#sec-15.12.2.

[2] Edge Device Characteristics - Akamai. https://learn.akamai.com/en-us/webhelp/ion/oca/GUID-8DC8807F-B65E-40EC-BB14-896C9F12026E.html.

[3] Fact & Figures - Akamai. https://www.akamai.com/us/en/about/facts-figures.jsp.

[4] ION Web Performance Optimization - Akamai. https://www.akamai.com/us/en/products/performance/web-performance-optimization.jsp.

[5] The json data interchange syntax. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[6] Kona Site Defender - Akamai. https://www.akamai.com/us/en/products/security/kona-site-defender.jsp.

[7] Media types - iana. https://www.iana.org/assignments/media-types/media-types.xhtml.

[8] Mime types - mdn. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types.

[9] Progressive web apps - the app shell model. https://developers.google.com/web/fundamentals/architecture/app-shell.

[10] Symmantec Sitereview. https://sitereview.bluecoat.com.

[11] User agent string database. http://www.useragentstring.com/.

[12] *Lecture on Ngrams and Backoff Models*, 2009. http://l2r.cs.uiuc.edu/~danr/Teaching/CS546-09/Lectures/Lec5-Stat-09-ext.pdf.

[13] Butkiewicz, M., Wang, D., Wu, Z., Madhyastha, H. V., and Sekar, V. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, May 2015), USENIX Association, pp. 439–453.

[14] Choi, B., Kim, J., Cho, D., Kim, S., and Han, D. Appx: an automated app acceleration framework for low latency mobile app. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies* (2018), ACM, pp. 27–40.

[15] Fielding, R., and Reschke, J. Rfc 7231-hypertext transfer protocol (http/1.1): Semantics and content. *Internet Engineering Task Force (IETF)* (2014). https://tools.ietf.org/html/rfc7231.

[16] Gerber, A., and Doverspike, R. Traffic types and growth in backbone networks. In *Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2011* (2011), Optical Society of America.

[17] Higgins, B. D., Flinn, J., Giuli, T. J., Noble, B., Peplin, C., and Watson, D. Informed mobile prefetching. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (2012), ACM, pp. 155–168.

[18] Kline, J., Barford, P., Cahn, A., and Sommers, J. On the structure and characteristics of user agent string. In *Proceedings of the 2017 Internet Measurement Conference* (New York, NY, USA, 2017), IMC '17, ACM, pp. 184–190.

[19] Miskovic, S., Lee, G. M., Liao, Y., and Baldi, M. Appprint: automatic fingerprinting of mobile applications in network traffic. In *International Conference on Passive and Active Network Measurement* (2015), Springer, pp. 57–69.

[20] Nejati, J., and Balasubramanian, A. An in-depth study of mobile browser performance. In *Proceedings of the 25th International Conference on World Wide Web* (2016), International World Wide Web Conferences Steering Committee, pp. 1305–1315.

[21] Netravali, R., and Mickens, J. Prophecy: Accelerating mobile page loads using final-state write logs. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (Renton, WA, Apr. 2018), USENIX Association, pp. 249–266.

[22] Nygren, E., Sitaraman, R. K., and Sun, J. The Akamai network: A platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 2–19.

[23] Poese, I., Frank, B., Smaragdakis, G., Uhlig, S., Feldmann, A., and Maggs, B. Enabling content-aware traffic engineering. *ACM SIGCOMM Computer Communication Review 42*, 5 (2012), 21–28.

[24] Pujol, E., Richter, P., Chandrasekaran, B., Smaragdakis, G., Feldmann, A., Maggs, B. M., and Ng, K.-C. Back-office web traffic on the internet. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (2014), ACM, pp. 257–270.

[25] Richter, P., Padmanabhan, R., Spring, N., Berger, A., and CLark, D. Advancing the Art of Internet Edge Outage Detection. In *Proceedings of ACM IMC 2018* (Boston, MA, November 2018).

[26] Richter, P., Padmanabhan, R., Spring, N., Berger, A., and Clark, D. Advancing the art of internet edge outage detection. In *Proceedings of the Internet Measurement Conference 2018* (2018), ACM, pp. 350–363.

[27] Sivakumar, A., Puzhavakath Narayanan, S., Gopalakrishnan, V., Lee, S., Rao, S., and Sen, S. Parcel: Proxy assisted browsing in cellular networks for energy and latency reduction. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2014), CoNEXT '14, ACM, pp. 325–336.

[28] Trevisan, M., Giordano, D., Drago, I., Mellia, M., and Munafo, M. Five years at the edge: watching internet from the isp network. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies* (2018), ACM, pp. 1–12.

[29] Vlachos, M., Yu, P., and Castelli, V. On periodicity detection and structural periodic similarity. In *Proceedings of the 2005 SIAM international conference on data mining* (2005), SIAM, pp. 449–460.

[30] Wang, X. S., Balasubramanian, A., Krishnamurthy, A., and Wetherall, D. Demystifying page load performance with WProf. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)* (Lombard, IL, 2013), USENIX, pp. 473–485.

[31] Wang, X. S., Krishnamurthy, A., and Wetherall, D. Speeding up web page loads with shandian. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, 2016), USENIX Association, pp. 109–122.

[32] Xu, Q., Andrews, T., Liao, Y., Miskovic, S., Mao, Z. M., Baldi, M., and Nucci, A. Flowr: a self-learning system for classifying mobileapplication traffic. *ACM SIGMETRICS Performance Evaluation Review 42*, 1 (2014), 569–570.

[33] Xu, Q., Erman, J., Gerber, A., Mao, Z., Pang, J., and Venkataraman, S. Identifying diverse usage behaviors of smartphone apps. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (2011), ACM, pp. 329–344.

[34] Yao, H., Ranjan, G., Tongaonkar, A., Liao, Y., and Mao, Z. M. Samples: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (2015), ACM, pp. 439–451.

[35] Zhao, Y., Laser, M. S., Lyu, Y., and Medvidovic, N. Leveraging program analysis to reduce user-perceived latency in mobile applications. In *Proceedings of the 40th International Conference on Software Engineering* (2018), ACM, pp. 176–186.