Leveraging your CDN to cache "uncacheable" content

Note: This blog post was originally published in July 2014 under the title "Leveraging your CDN to cache dynamic content." In light of my article about the rise of event-driven content, I decided that this blog post needed updating. Below, I'll clarify the different content classes and discuss why CDNs can (or can't) cache them.

For years, traditional CDNs have taught us that any content that requires application logic (like HTML pages, API calls, or AJAX requests) is "dynamic" and therefore uncacheable. As applications have gotten smarter and deployed more logic, more and more content has been classified under this category. This makes it difficult for companies to deliver content in real time to their users without sacrificing performance. Over time, this approach has caused caching strategies to suffer and lessened the impact a content delivery network can have on efficient content delivery.

The truth is that not all of this content is truly dynamic and uncacheable. As I've written about before, there is a growing class of content called "event-driven content" that may appear dynamic, but isn't. Even though caching this type of content is more difficult, it's possible to cache it with the right tools — although you might have been told otherwise. In doing so, you can leverage the power of caching in new ways and reap the benefits that come along with it.

Dynamic vs. event-driven content

Traditionally, our industry has used the word "dynamic" to define content that isn't static. Many people in the CDN world have used this term to refer to any type of content that is generated "dynamically," through the use of some sort of application logic.

A lot of your content falls under this unnecessarily broad definition, the most typical of which is the base HTML on a web page. In web apps, HTML is where an application does its thinking, generating a page that the browser can then render. Base HTML is a big deal in the web performance community because it's a blocking request, meaning that nothing can be downloaded by the browser until the HTML starts arriving. It's also often larger, so it needs time to fully get to the browser.

But it's not just HTML that qualifies as "dynamic" under this definition. For example, AJAX requests and API calls would also fall into this traditionally broad category.

As I've discussed previously, using the word "dynamic" to define everything that isn't static and predictably cacheable is not really accurate. Dynamic content should refer to objects that are truly dynamic — objects that are never the same twice. Requests for dynamic content must go to origin every time. In other words, dynamic content is always uncacheable.

Event-driven content looks like dynamic content, but it really isn't. What sets event-driven content apart from dynamic content is that it's actually static, but for unpredictable periods of time. This means that you can't determine cacheability lifetime in advance. In web applications, for example, most HTML content is actually event-driven rather than truly dynamic. For further reading, check out my post about event-driven content.

Because this class of content is difficult to cache, the CDN industry has called it dynamic and labeled it uncacheable for years. However, event-driven content is very cacheable — a CDN just needs to have the right mechanisms in place to cache it.

Why is caching event-driven content difficult?

Most CDNs handle event-driven content the same way they handle dynamic content: with a no-caching approach. There are two major motivations for this:

1. Lack of a good invalidation framework

HTTP gives us a lot of guidance and rules around caching. Chapter 13 of RFC 2616 provides a lot of knobs and levers in terms of headers for caching objects. However, the RFC gives very little guidance around invalidating cached content, with no standardized mechanisms for getting things out of a cache if you incorrectly determine (or, rather, guess) the expiry time of an object through Cache-Control headers.

Many businesses have an extreme fear of serving stale content to their users. Imagine a media site that publishes a news article, caches it in the network, realizes that the reporter got a fact wrong, and then has no way to retract the article until it expires from the cache. (By the way, that's an actual story that someone told me at a conference.) Without a proper invalidation framework, the unpredictability of event-driven content has been motivation enough to avoid caching it.

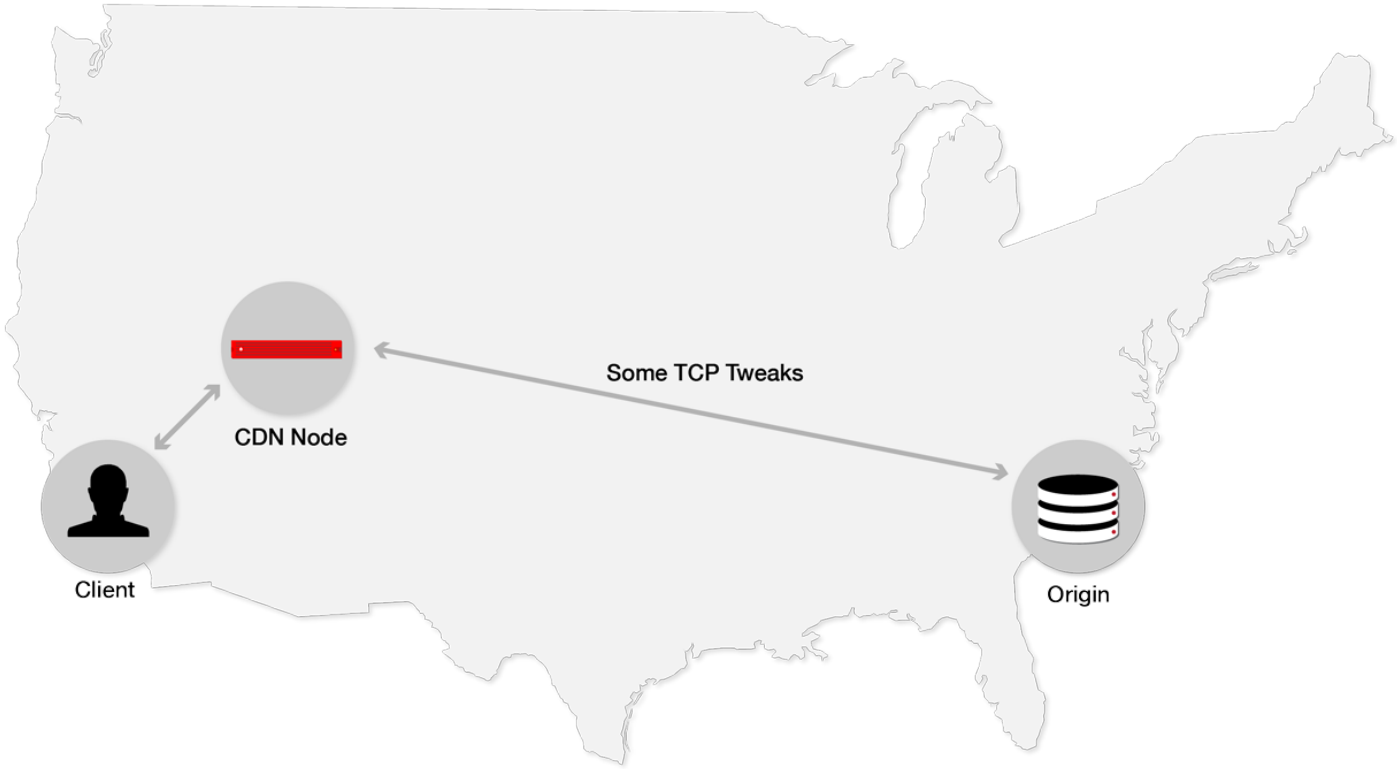
2. Lack of visibility

Another challenge that often prevents website owners from caching event-driven content is that putting that content on a CDN will make them lose some visibility into site traffic. Many analytics and stats mechanisms are based on collecting logs of user visits on an origin infrastructure. Putting these types of objects on a traditional CDN means that you won't see the hits or log the visits, causing you to lose valuable insight into how your users interact with your applications.

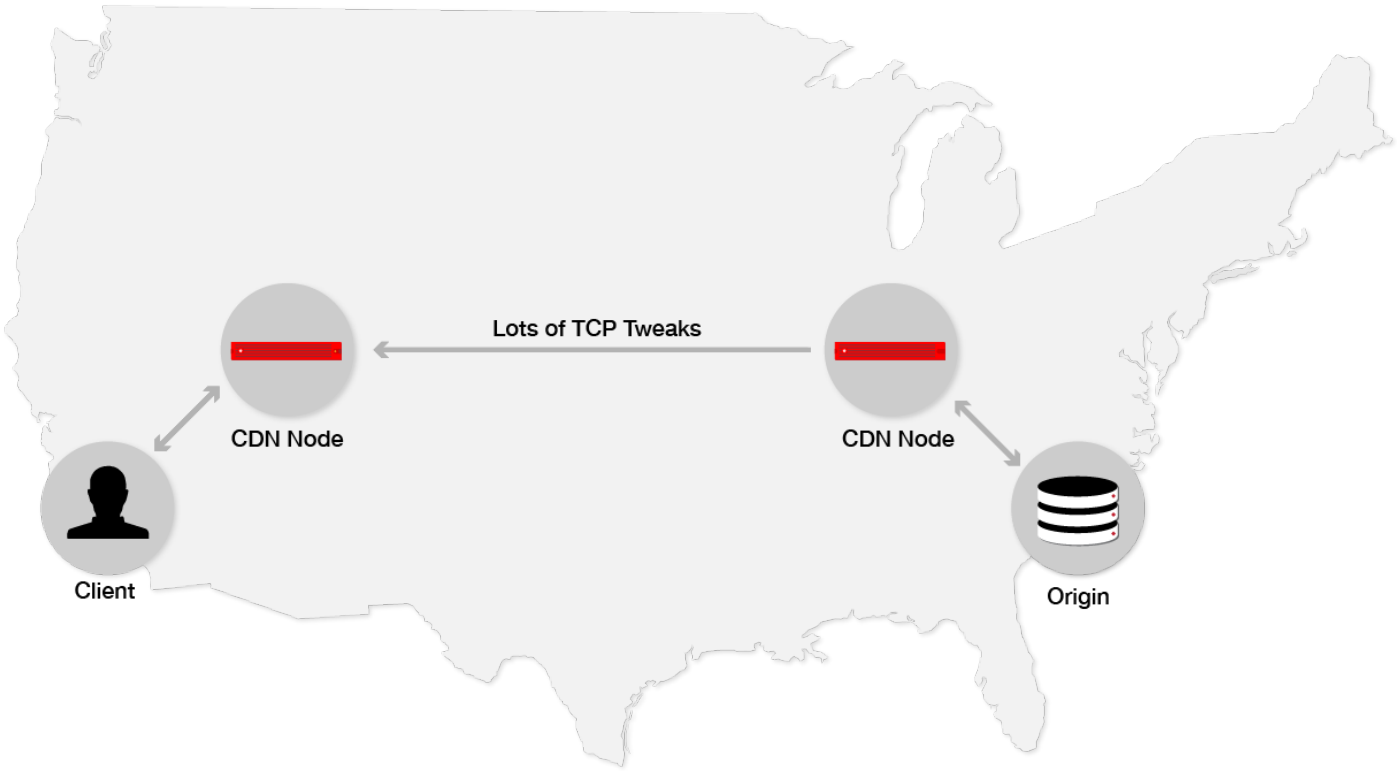
The classic approach towards event-driven content delivery

Most CDNs have treated event-driven content as dynamic and focused their attention on fast delivery of these types of objects from origin. Technologies like Dynamic Site Acceleration (DSA) focus on getting the content from origin to the client as fast as possible. Using TCP optimizations from edge nodes to origins, some of the networking overhead is mitigated and content is delivered more quickly from origin to edge and then to the client. This results in an asymmetric optimization of the delivery path.

Some CDNs also offer additional services where requests and responses are routed symmetrically from the CDN node closest to the client through a CDN node closest to the origin, and vice versa. This means the longest path (the "middle mile") is between the two CDN nodes, where lots of TCP optimizations can guarantee the best network path across the farthest part of the delivery route.



Traditional asymmetric optimization for dynamic and event-driven content delivery from origin



Traditional symmetric optimization for dynamic and event-driven content delivery from origin

To be fair, event-driven content hasn't always been delivered from origin by traditional CDNs. In cases where you know approximately how long your event-driven content can be cached, you can still use Cache-Control headers. In these cases, traditional CDNs have prescribed using really short cache expiry times for this content (in the order of minutes), while marking your static content as cacheable for much longer periods of time (weeks or months).

With traditional CDNs, purging content is not instantaneous, often taking minutes or even hours. This means you're left at the mercy of your cache expiry headers (or preset configuration for default cache expiry times), and any mistakes you make in content publishing can end up making things worse. Because of this, you may have to be extremely conservative when choosing to cache event-driven content on a more traditional CDN.

Using the low-expiry approach has never been an optimal caching strategy since you sacrifice good caching to hopefully minimize how long you may serve stale content. (Also, your fingers probably really hurt from always keeping them crossed, since choosing these low expiry times is essentially an exercise in your tolerance for mistakes.)

Some sites also use Edge Side Includes (ESI) with the goal of cutting up an HTML page into fragments, where each piece has its own independent caching heuristics. A CDN can cache the static portions of the template and call back to origin for the non-static content, assembling the page at the edge and then delivering the page in its entirety to the client. ESI is a very valid technique (we support ESI at Fastly), but developers often find it hard to implement. Plus, it requires lots of extra code in HTML pages.

Fastly's approach to event-driven content

There are major benefits to caching event-driven content. For standard web traffic, pages will get delivered faster since the HTML is often served from a node closer to the actual client than the origin servers. Additionally, the offload benefits that caching provides for your origin are even more appealing when it comes to event-driven content. This is largely because caching can save you a lot of compute time in your application, not to mention all the requests that no longer need to go to origin.

Recognizing these benefits, Fastly has fundamentally addressed the two major problems with caching event-driven content:

1. Invalidation through Instant Purge

To address the unpredictable nature of event-driven content, Fastly provides Instant Purge, an instantaneous invalidation mechanism which removes the burden of requiring predictable expiry for good caching. With Fastly, you don't need to rely solely on headers or configuration settings to determine how long content can be cached. Event-driven content can be cached as needed, and it can be instantly purged whenever an invalidation event occurs. All of this happens through the Fastly API, in ~150 milliseconds.

Every CDN has a purging system, but that system can only be used for caching this type of content if the invalidation happens instantly. With Instant Purge, there is no longer a need to guess at expiry times for event-driven content in advance. You can just purge as content becomes no longer valid.

Fastly also lets you take Instant Purge even further. Using surrogate keys, you can create dependencies and purge entire groups of objects together through unified tags. For more on surrogate keys, check out parts one and two of our blog series on the subject.

Instant Purge puts the control back in your hands and lets you publish and invalidate content as needed. Naturally, this feature is quite popular with our customers. Many have built it into their CMS and development environments. We've also published plugins for Wordpress, Drupal, and Ruby on Rails.

2. Real-time logging and visibility

Fastly has addressed the visibility issues around caching event-driven content through the implementation of Real-time Analytics and Real-time Log Streaming.

Stats and analytics are available in real time through the Fastly Dashboard and through our API. Both are popular with our customers; many use our Dashboard as a core part of their own application monitoring systems.

Fastly's Real-time Log Streaming can stream W3C HTTP logs instantly to any number of log destinations (syslog, S3, FTP, etc.), and is compatible with logging-as-a-service providers like Splunk, Logentries, and Papertrail. Varnish variables can also be added to the log entries, providing even more insight into user traffic. And, as with most Fastly features, logging is conditional. In other words, rules can be set up to stream logs only when certain traffic criteria is met. Here's a good write-up on some of Fastly's logging capabilities.

What about dynamic content?

We've put a lot of focus on event-driven content, but it's important to note that we haven't forgotten dynamic content. To reiterate, this type of content is never the same twice and can never be cached.

Because Fastly's customers rely on us to deliver all types of content — including dynamic content — DSA is an important part of our platform. As such, we've optimized delivery from origin over long distances. Through the use of network and protocol

optimization, along with mechanisms such as Origin Shield, Fastly's DSA offering is capable of transporting truly dynamic content over long distances and with efficient speed.

Final thoughts

Our Instant Purge and real-time visibility features help businesses cache a lot of content that would otherwise be uncachable with traditional CDNs. These mechanisms are powerful enablers for content delivery, and help multiple facets of our customers' applications, including performance, origin offload, and overall insight.

For further reading, check out these two articles about ways Fastly customers have cached content that would have otherwise been considered uncacheable: "API caching" and "How to cache with tracking cookies."

PERFORMANCE

Published May 5, 2015

Want to continue the conversation?

Schedule time with an expert

Share this post

Hooman Beheshti
VP of Technology

Hooman Beheshti is VP of Technology at Fastly, where he develops web performance services. A pioneer in the application acceleration space, Hooman helped design one of the original load balancers while at Radware and has held senior technology positions with Strangeloop Networks and Crescendo Networks. He's been developing the core technologies that make the Internet work faster for nearly 20 years and is an expert and frequent speaker on the subjects of load balancing, application performance, and content delivery networks.

Subscribe to our newsletter

Get the latest news and industry insights in your inbox.

Work Email
Enter your email address

Submit