Leaving behind
ADC & FEO:

A CASE STUDY IN "CONTENT ORCHESTRATION" FOR APP ACCELERATION

YOTTAA

Contents

Introduction

Prevailing wisdom in web technology holds that in order to achieve faster performance, applications must be pared down. Apps today are heavy and bloated, the idea goes, and "lighter and simpler" is the prudent technologist's new mantra.

We'll admit it's a pleasantly intuitive idea. While internet connections are getting faster and hardware is constantly improving, web apps are actually getting *slower* – so something is clearly wrong. And when looking for the source of this dissonance, the obvious culprits are high-resolution images, JavaScript-enabled features, and third-party ads, trackers and scripts. This proliferation of rich content has led to the average web app growing threefold in weight since 2012; it makes sense to blame it for the problem of widespread slow performance. This assumption, however, misses a bigger problem.

In our view, it's **the means by which we approach optimization** that has veered off course, not the contents of the apps themselves. While the creators of web apps are not entirely blameless, they've largely followed a natural progression in adding functionality and fidelity. In contrast, methods of application delivery and optimization available in the market have scarcely evolved. Applying these outdated optimization techniques to today's complex applications has left users frustrated, wasted development resources, and burned technology budgets with negative-ROI solutions. This is the real problem, and its only viable solution is to modernize our approach to optimization.

To prove our thesis we'll show that it's possible to use next-generation "content orchestration" techniques to create a brilliant user experience for all users with an application that on paper should be bloated and slow. And we'll do it without reducing functionality.

But first, some background. **What's the current state of optimization?**

Thesis

According to those "in the know" in internet technology, there are two primary mechanisms to accelerate today's dynamic, Internet-facing applications: **network optimization and front-end optimization (FEO)**. In fact, these are the only separate categories listed in Gartner's Technology Insight for Content Delivery Networks (aside from edge caching, which is the base feature of a CDN).

While this reading of the market has generally held true for the past decade, lately some of the inherent limitations in each of these approaches have become evident.

For one, **network optimizations** are up against the laws of physics. Where dynamic/personalized content is concerned, the bytes must always traverse a path from the visitor's device through the Internet to an origin server, where the content is generated by a database, then make the trip back again. In other words, no amount of route optimization or cache management can change the fact that some significant geographic distance must be covered by the data.

As a result, online applications that rely on pure CDN and ADN technologies are getting slower: the amount of dynamic content they host is growing quickly, while latency can only be reduced by tiny increments, if at all.

FEO, for its part, suffers from being both too narrow and too broad. On one hand, it is limited by a scope that begins and ends with content that is owned and managed by the application owner and can be re-written or re-organized manually. In a world where third party content is taking an increasing share of applications, this is a problem. On the other hand it's overly broad in the variety of techniques that fall under the moniker. Official descriptions of FEO include one-off programming hacks, software automation (much of which has since migrated to browsers), and complex feats of application engineering – all rolled into one:

> *For the true "last mile," web performance optimization — often called front-end optimization (FEO) — manipulates assets to optimize time-to-first-render and time-to-first-interaction. Web performance optimization solutions achieve these feats by: 1) bundling assets and images to make fewer round trip calls; 2) sequencing elements, e.g., focusing on above the fold content first or progressively painting content; 3) compressing code and images; and 4) image format, resolution, and art direction changes.*
>
> *Market Overview: CDN Platforms and Digital Performance Services* (Forrester Research)

> *Front-end optimization (FEO), sometimes called Web content optimization or Web performance optimization, alters the pages produced by the Web servers, so that they are both delivered faster by the network and rendered more quickly by the browser. Examples of FEO techniques include: Combining multiple Cascading Style Sheets (CSS) files into a single file. Combining multiple images into a single file, using CSS sprites and inlining…Increasing cache efficiency by manipulating headers, such as altering expiration times. Sending only deltas (changes), if the client has retrieved the page before. Taking advantage of the application cache in browsers that support HTML 5.*
>
> *Technology Insight for Content Delivery Networks* (Gartner)

Analyst-speak aside, "on the ground" industry sentiment says that FEO as a whole is brittle – i.e., it frequently causes errors and requires more time than it's worth. What's more, it will generally be made redundant with a broad adoption of HTTP/2. For example, the following FEO optimizations, already marginalized, would further fade to insignificance with HTTP/2:

- Script combination
- Script inlining
- Script minification
- Domain sharding/request parallelization

Other key FEO optimizations might still "work" in the strictest sense in an HTTP/2 world, but will have a diminishing impact over time. These include:

- GZip compression
- DataURI scheme
- Lossless & lossy image compression
- Image right-sizing
- Image transcoding
- Responsive imaging

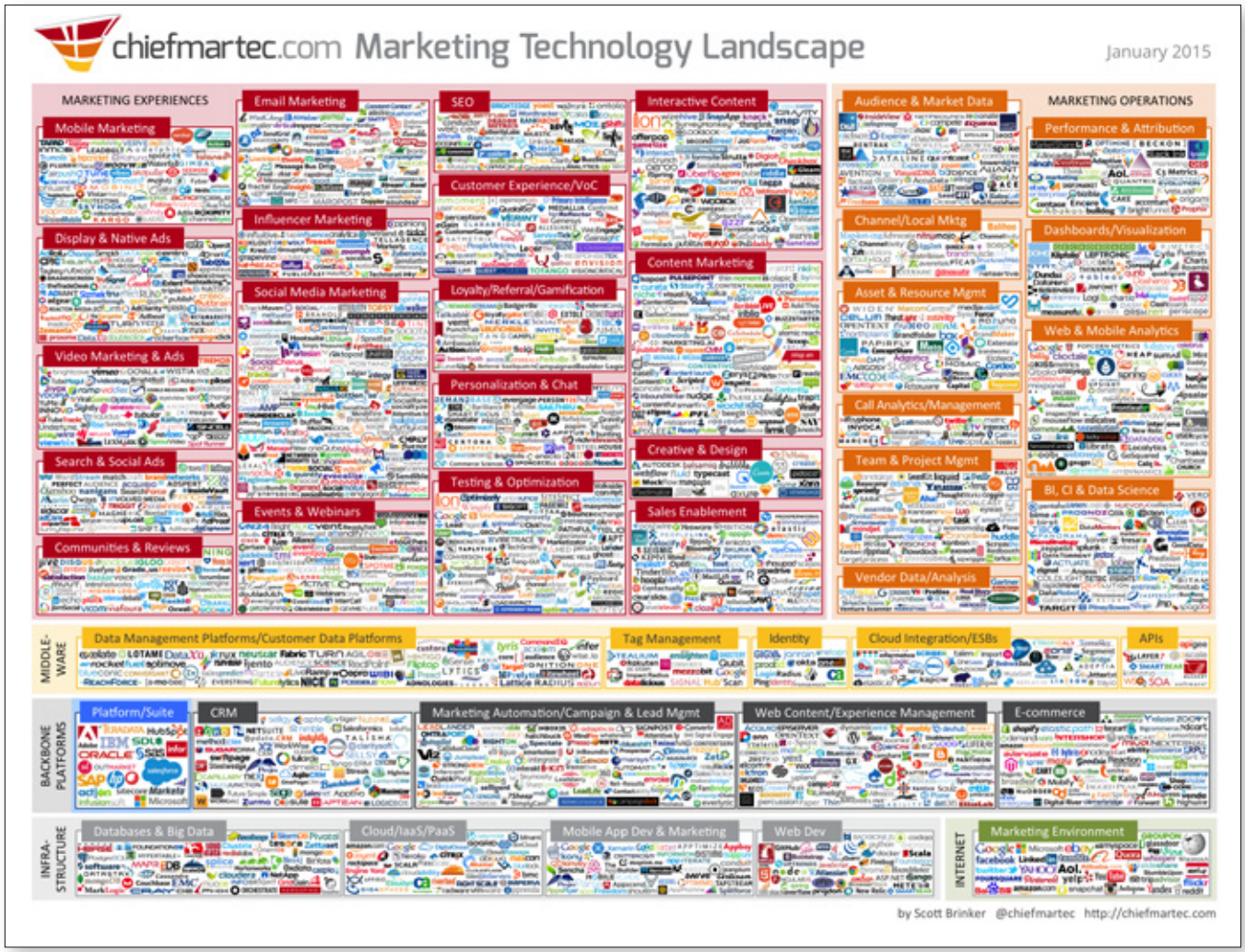
These techniques help to cut down the number of bytes, particularly for images, but don't account for dynamic content or third party content. That would make them what we call "technical gravy": useful in certain cases as a point solution, but otherwise irrelevant to overall user experiences.

You may have picked up on a theme here. The biggest problem with legacy optimization methods is the **inability to address dynamic, personalized, and third party content.** To make modern applications fast, you simply can't avoid the source of the problem.

The Modern Application

To find out how to approach the source of the problem, we need to understand the applications we're dealing with.

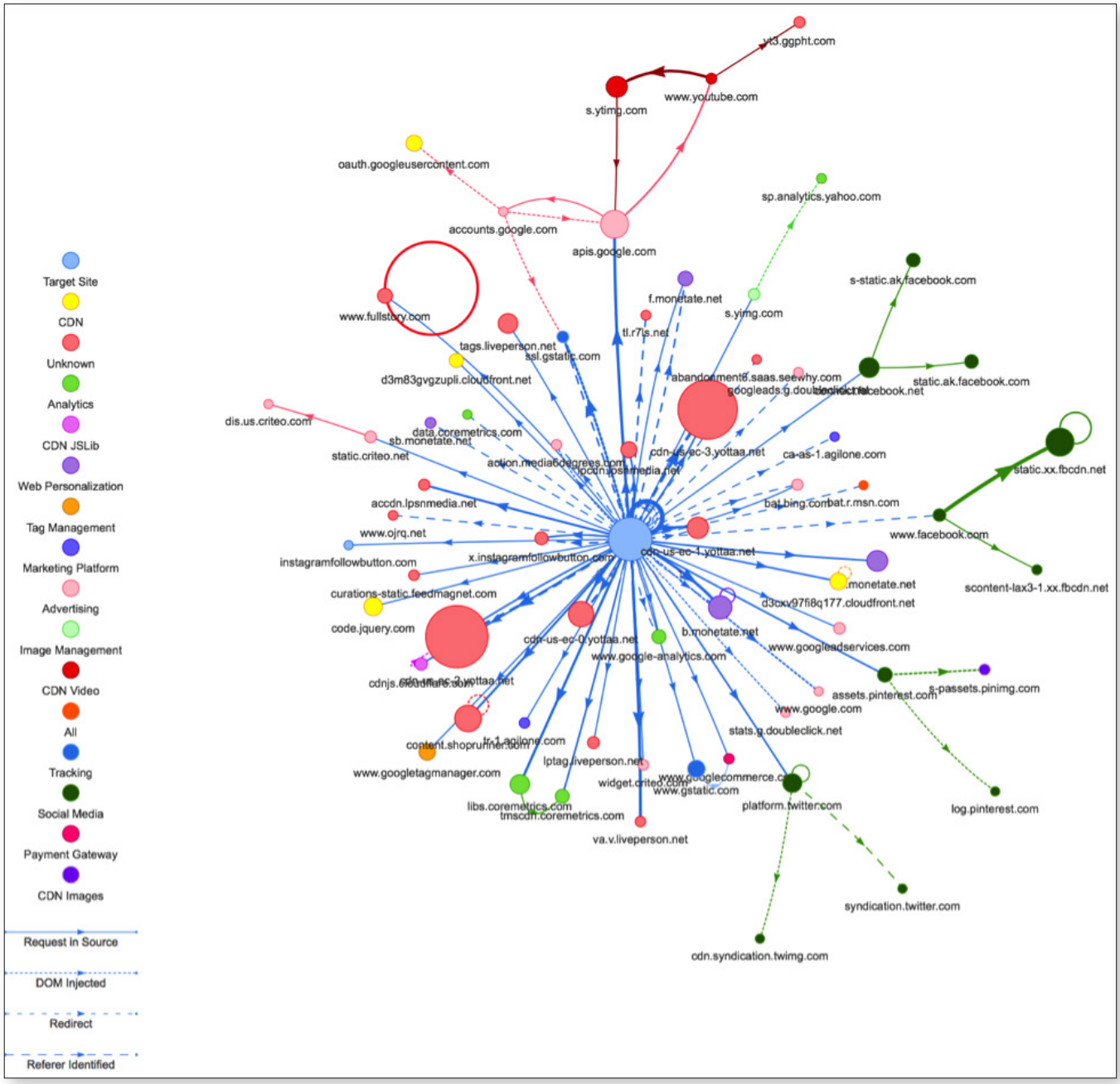
Dynamic applications come in all shapes and sizes, and some of the most complex are eCommerce portals. Aside from dealing with how to represent and organize thousands, hundreds of thousands, or even millions of SKUs, applications must also account for users on a variety of devices and the expectation for personalized features. This scenario has given rise to a dizzying array of tools that promise to draw the sword (revenue) from the stone (fickle consumers). They invariably operate by inserting "just one innocuous piece of JavaScript" to your application.



(Click to enlarge)

Looking at this staggeringly rich ecosystem, one might begin to question the default wisdom of prioritizing "simple, lightweight" applications. If your competitors are freely leveraging a bevy of such tools, how can you avoid creating your own cattle car of distributed domain calls to these services? How do you say no to a tool that might give you even the slightest edge?

For an example of how these questions often play out, let's look at a popular eCommerce portal for a sports equipment retailer. This is a *request map* showing how content is being added to the site:



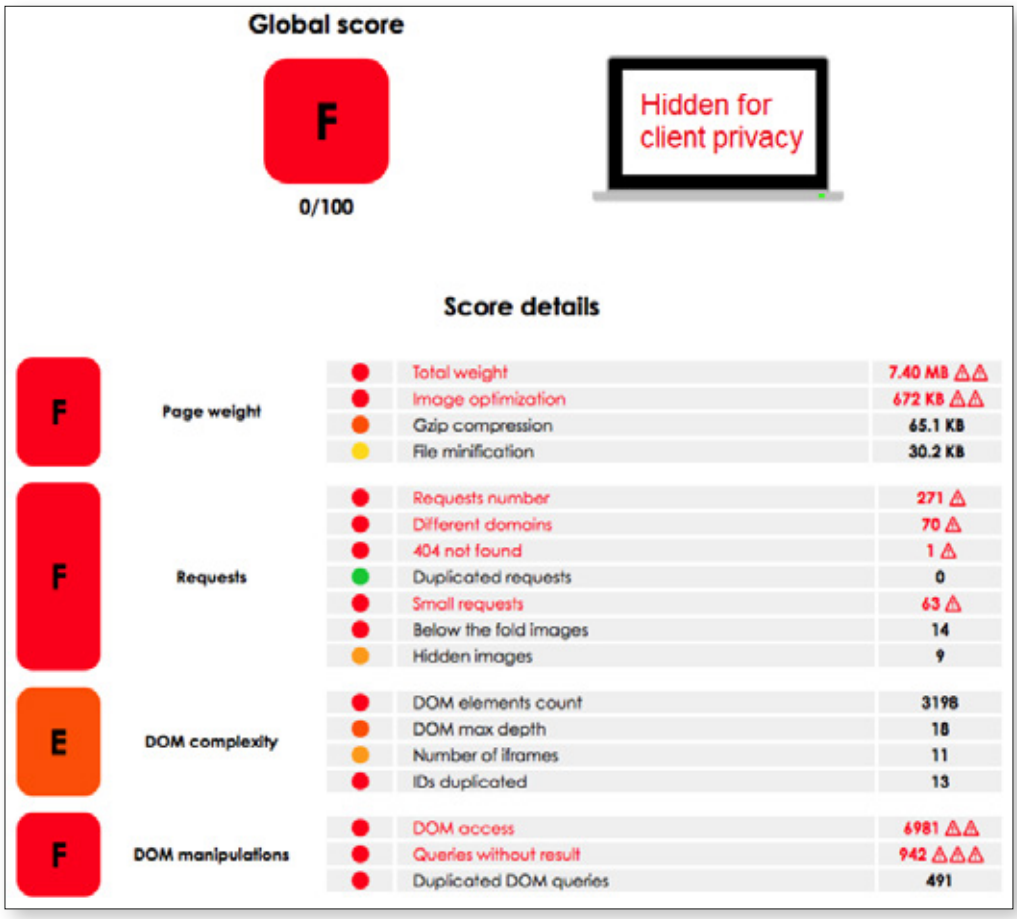
Not for the faint of heart, what this tangled mess represents is:

- Nodes for every separate domain from which content is being fetched
- The volume (bytes) of content being retrieved, represented by the size of the node
- The amount of latency inherent in retrieving that content, represented by the length of the lines

The chart serves to visualize just how much is happening behind the scenes to produce a conventional eCommerce portal today. Dozens of separate domains serve hundreds of calls for content, some even begetting further calls to tertiary sites. For the question of "how to say no" to 3rd party tools the answer is, at least for this company, "we don't." The chart also shows the extent to which a typical site's content is completely outside of a CDN or FEO implementation's control.

So how does an application with this level of complexity map out in typical measures of performance, based on the status-quo assumptions?

According to a web performance grader tool from *Yellow Labs*, it scores very poorly indeed. This tool takes a number of "best practices" that, in theory, lead to great user experiences, and scans any site to see how many it meets. Our case study eCommerce app receives an unequivocally failing grade: zero out of 100.



It would be easy to write this off as a poorly made site, run by idiots, and sure to fail. Unfortunately, we know better: that it's a longtime member of the Internet Retailer 500, the list of the top-grossing eCommerce firms in the country. Slow and technically indebted it may be, but it's not failing.

How does a successful site get to this point? Let's review some of the content on this page by way of a thought exercise. The following are a sample of the scripts that are present on the page and contributing to the failing grade. **What would you remove?**

- **Coremetrics** - provides business insight so site owners can make intelligent decisions
- **Monetate** - optimizes the visitor's journey with targeted, personalized content
- **LivePerson** - allows shoppers to get real-time help with their journey
- **Criteo** - enables retargeting which is proven to optimize conversion rates by following users around the internet with reminders that they really want that hoodie they were browsing for...

This exercise brings into focus the tension between eager marketers and concerned technologists. Clearly, each of these tools was implemented with specific intentions and ROI goals. Removing any one could cut into customer acquisition and retention efforts, leading to a potential loss of market share in a brutally competitive online retail environment. And yet their cumulative effect on performance is such that it cuts into the user experience and drags down the effectiveness of all aspects of the site. It looks like we're stuck.

But here's where the status-quo assumptions break down. As noted above, conventional wisdom holds that to improve performance for an application of this breadth and scale, a good number of these features must be cut entirely, and a battery of time consuming front-end optimizations must be implemented, likely by a dedicated and skilled performance team. Not so. What follows are results taken after optimizing the same site we've examined here, which was accomplished with **zero changes to code, and no reduction in complexity or functionality**, through a method we call content orchestration.

Trend Watch: **Single Page Apps**

There's a growing revolution in how web applications are built to serve the needs of complex functionality. "Single page apps" (SPAs) built with frameworks like Backbone.js and Angular.js are different animals than traditional web apps – they're much less dependent on back-end logic, providing rich interactivity on the client side without having to make multiple calls for separate HTML pages to accomplish a task. Examples of such apps include Airbnb, Soundcloud, TurboTax, Google Analytics, airline websites such as Jetblue, and many many more.

These developments are a huge step forward in bringing the kind of responsiveness normally reserved for native applications to the web. Indeed from a developer's perspective, SPAs are more like desktop apps than web apps – as one blogger noted, "It means we can build real software in a web page instead of just using JavaScript for simple event handling and DOM manipulation."

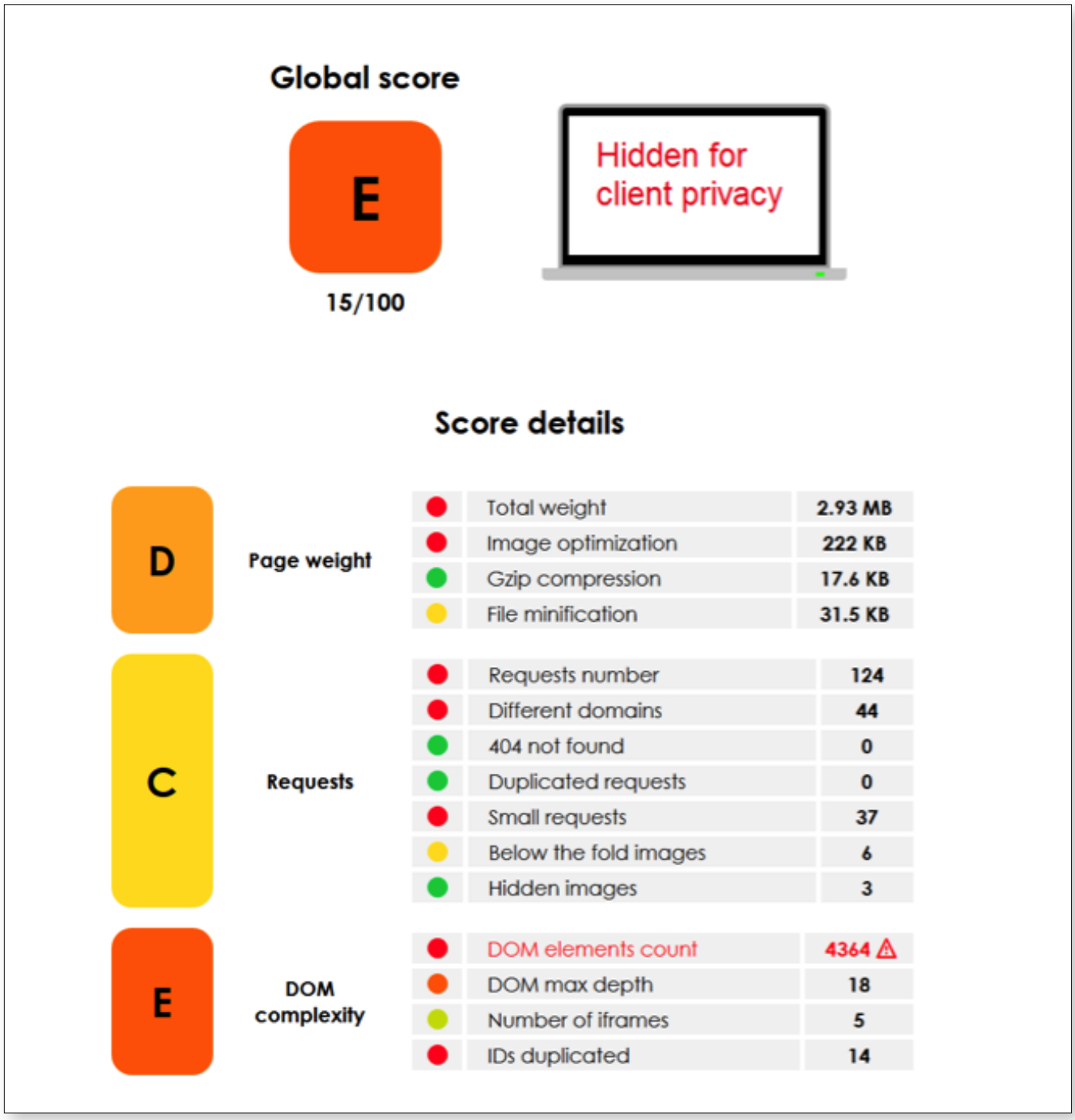
But as always, there's a catch. Single page apps like these might be built differently than traditional web pages, but they're the same in one way: the need to be extended via SaaS and third-party JavaScript. Just because a team builds a great ticket booking application with a current-gen framework doesn't mean they're also going to custom-code an analytics platform or an advertising network. Those services will still execute the same way they do sites like the one exemplified in this document – by calling external scripts and executing them on the page.

Single page apps and their ilk solve one big challenge – the ability to execute complicated tasks on the open web quickly and efficiently. But they can't solve the performance problems that plague the web. For that, we'll need a new paradigm for optimization, just as we have a new paradigm for app development.

Anatomy of an "Orchestrated" eCommerce Site

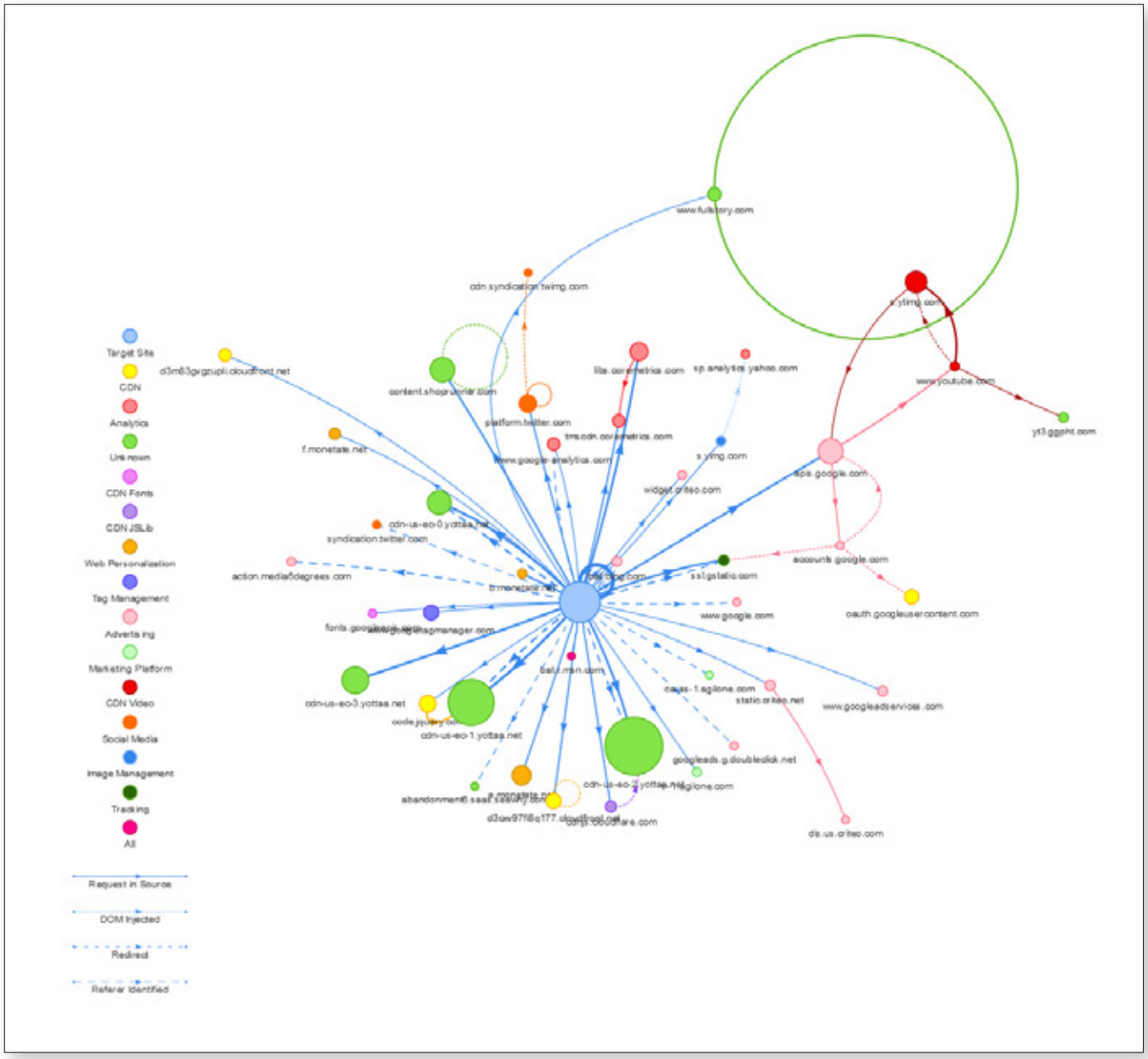
The "version" of the site we're looking at in this section is actually the same exact site, at least from the perspective of the codebase that the company maintains. The only thing that's changed is how it's delivered and optimized on the fly, once a request has been made.

Here's the performance grader score again. **Look for the differences:**



It's still terrible! Some minor improvements have been made but it's still just a 15/100 grade. The biggest difference comes in the reduction of the number of overall requests and separate domains, which are each cut roughly in half. More on that later.

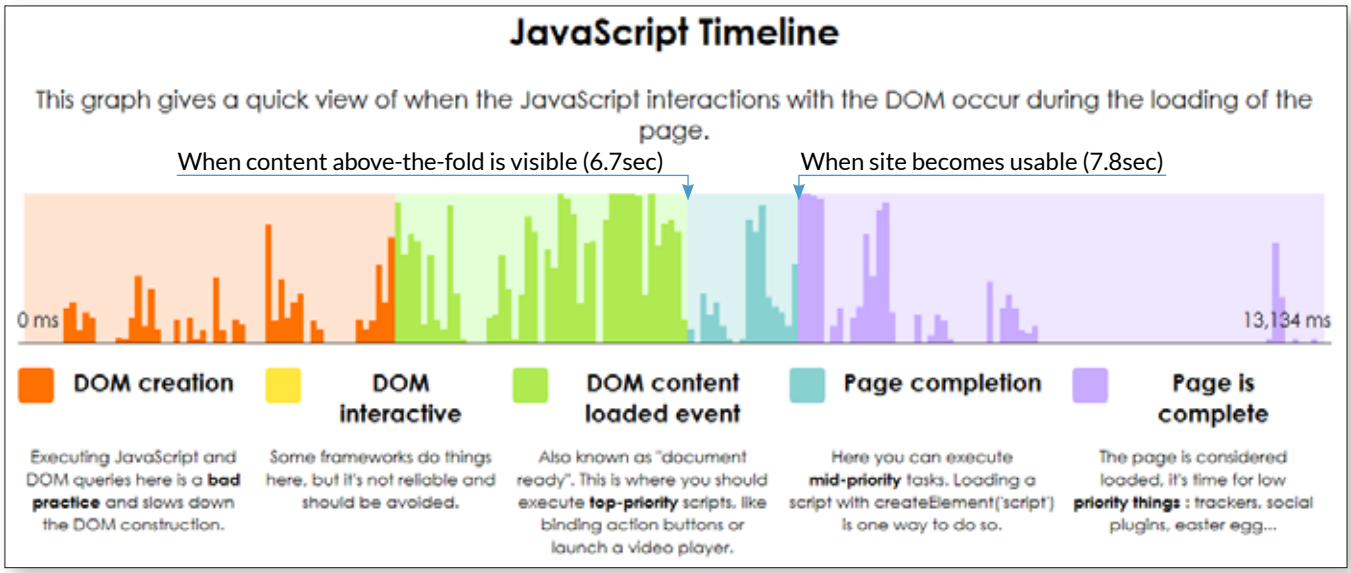
And here's the request map:



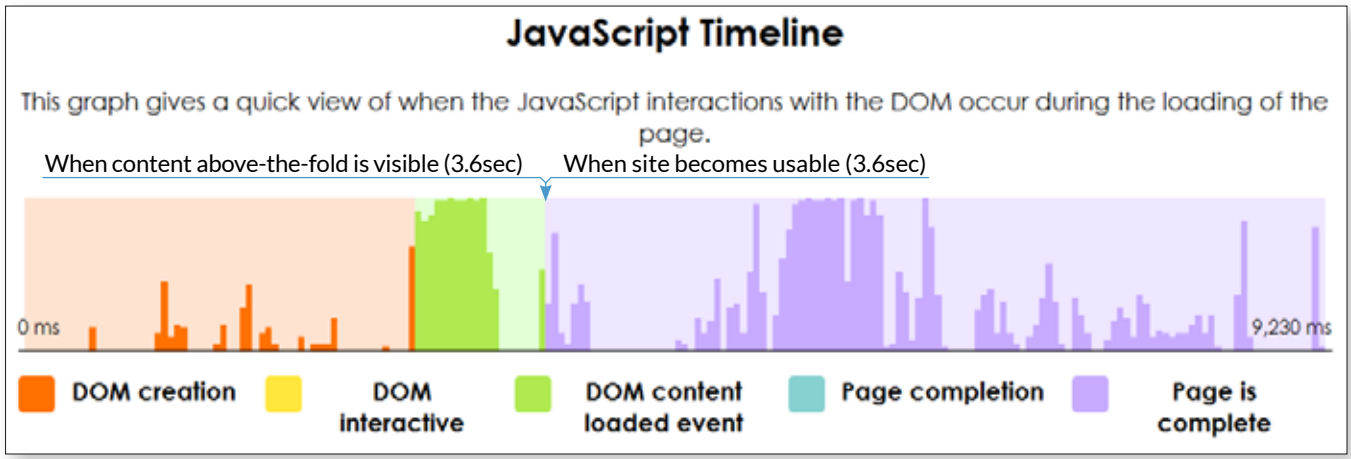
A bit more tame, but still wild looking, and with lots of latency involved. So far, little has changed.

Now things start to get interesting. Below are two timelines showing when JavaScript elements were loaded in relation to the overall timeline of the page loading. The first corresponds to the same "unoptimized" version that was on display in the first section, the second is the Yottaa-optimized version.

Original:



Optimized:



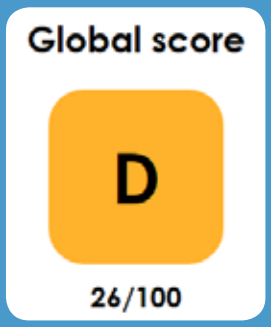
To the untrained eye, these might look like two sides of the same coin. On the contrary, what's happened in the process of optimization is the complete elimination of the "page completion" stage (teal), and a significant compression of the remaining stages. What that means in practice: not only is the crucial above-the-fold content fully rendered 3 seconds faster, the page becomes sufficiently complete as to be usable at the same moment – cutting off over 4 seconds of waiting. In an ADC/FEO world where shaving milliseconds of load time is the rule, this is a serious coup.

This result is accomplished by "breaking the rendering path" – that is, overriding the browser's normal behavior to skip or rearrange steps in the process. It's a natural byproduct of using content orchestration. Notice that in the first timeline, spikes of JavaScript load activity are distributed throughout 4 stages of the load process. In the second, much of the JavaScript action has shifted until after the point where page is usable. By sending static content from cache and re-prioritizing the loading of JavaScript, the overall experience has improved significantly.

Perception is Reality

The visualization also taps into a key concept: perceived speed is more important than actual speed. As you may have noticed, the difference between the technical "page complete" time has only moved from 13 seconds to 9 seconds. But what you don't know can't hurt you, as they say – and if the user gets served all of the key content they're looking for quickly (in 4 seconds or less) they'll happily engage, while other scripts can continue loading. This background loading can occur for as long as it needs to, provided the scripts don't interrupt the experience.

Think our masked example might be an outlier? We don't blame you. For proof of how common it is for eCommerce sites to diverge from performance best practices, consider the industry king: *Amazon.com*. Their *grader score* and *timeline* illustrate the same principles we're discussing. Amazon certainly doesn't get a "D" in revenue or market share.



And it's not just that the script processing has been pushed later. **Many elements have not been loaded at all**, a change we noted when looking at the website grader. How can that square with our claim that nothing on the site has been changed?

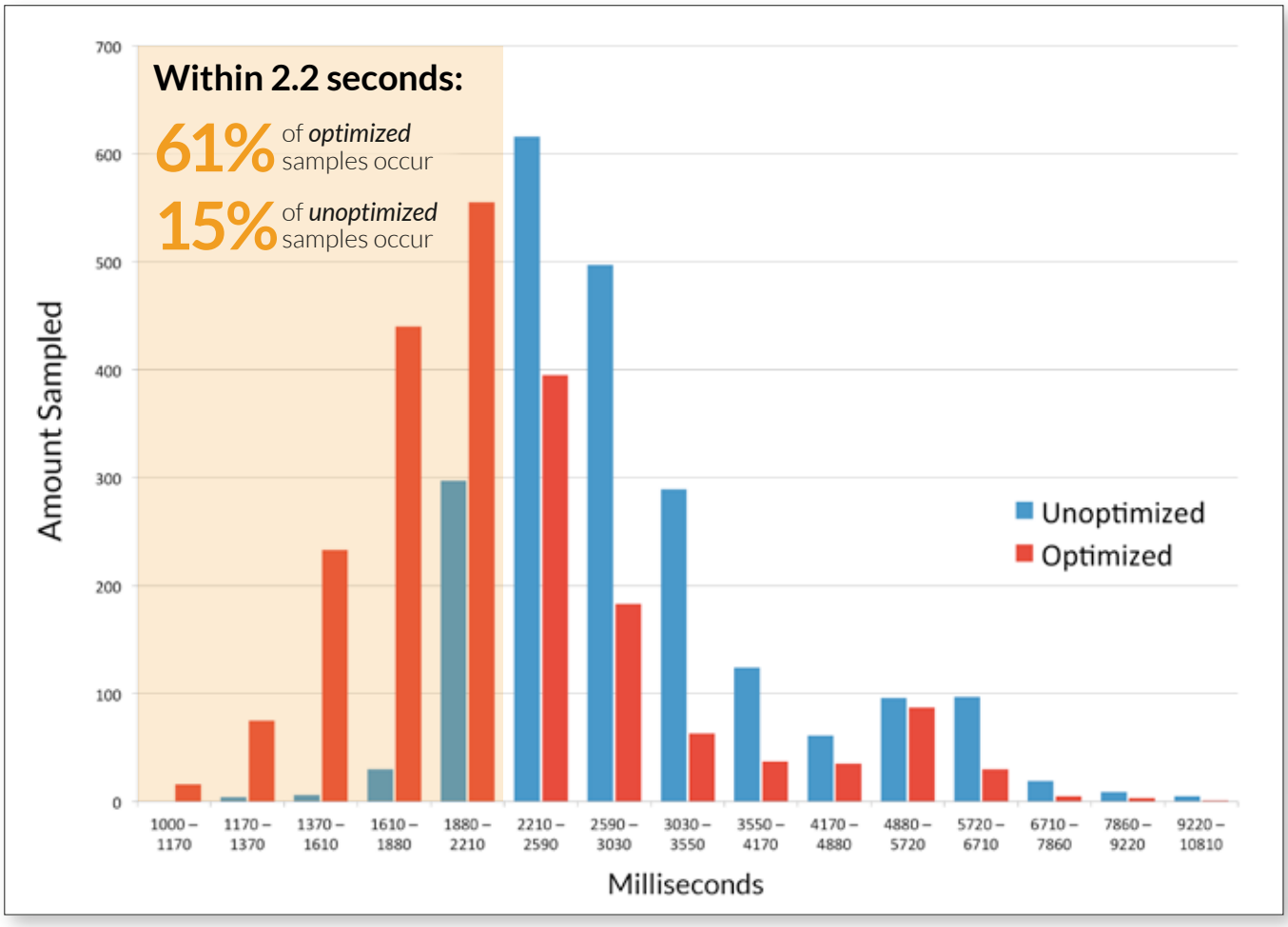
In fact, it's all still there, it's just waiting in the wings. In addition to the intentional delay of JavaScript, a number of elements have also been set to load only upon request – that is, when a user scrolls into the vicinity of the element or activates it with a hover or click. That means the initial view is rendered as if the site is lighter than it really is, without losing the overall functionality. By layering in content "just in time" with orchestration, we can further amplify the user experience.

A Customer-Focused Test

We've reviewed a single sample, which showed some compelling results. But a single sample is not the same as a consistent user experience. We have two other ways to quantify the improvements gained from an orchestration approach.

First – Distribution.

As you can see below, there's a pretty broad range for load times (in this case "document complete" time), optimized or not. The web is still an inconsistent and variable beast – you can't expect every user to get the same experience every time.

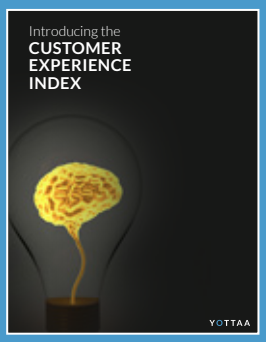


You can, however, minimize the variation as much as possible. The standard deviation between the optimized and unoptimized versions shown here has been reduced from 1.5 seconds to 1 second, while the average has gone from 3.1s to 2.3s.

As you'd expect based on those shifts, benchmarks like the 75th and 95th percentiles have undergone still more dramatic improvements. (Using high percentile benchmarks is a common practice in performance optimization to gauge variability). Not shown in the chart is a long tail of sample bins going up to 24 seconds, where only "Unoptimized" samples are present.

Second – Customer Experience Index

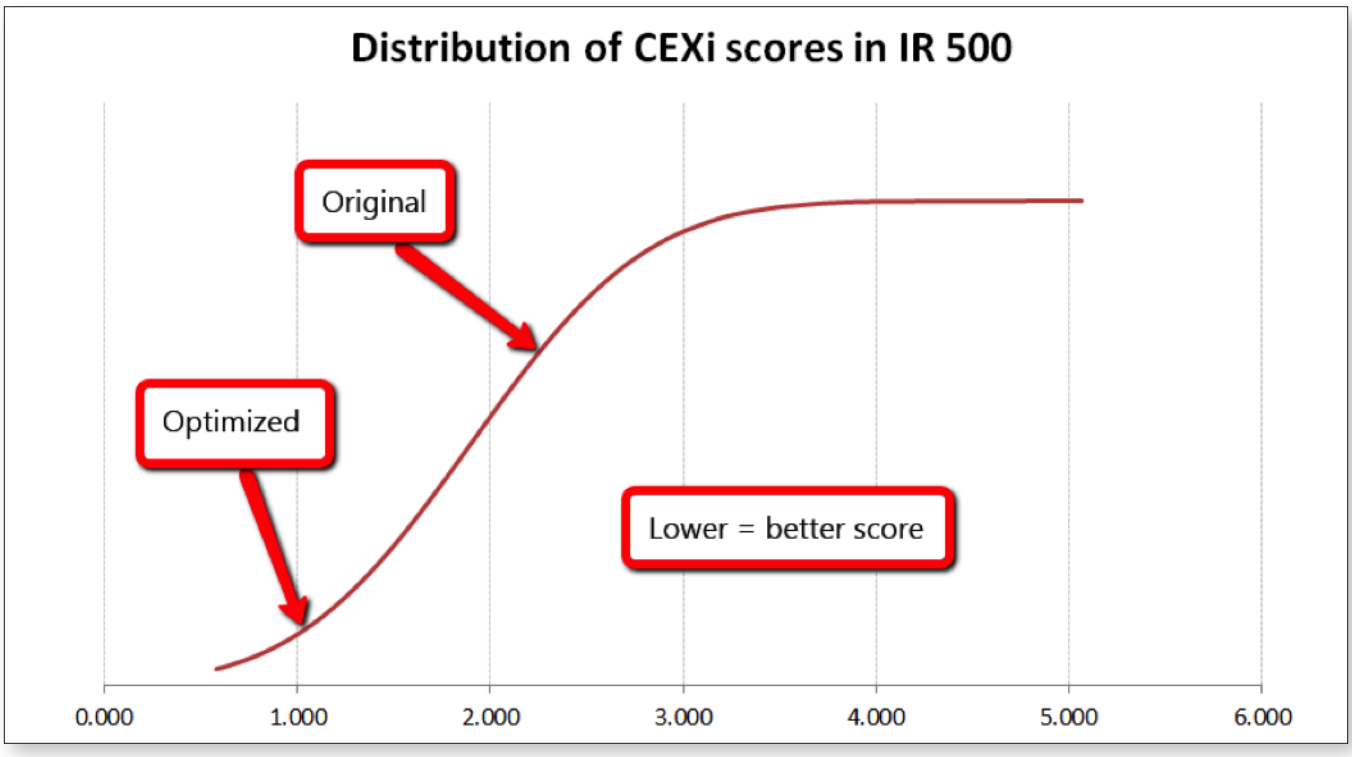
You can read much more about the CEXi in our intro guide.



The CEXi compiles conventional web performance measures like Time to Start Render and Time to Display for both mobile and desktop tests, and mixes in some extra weighting and balancing to account for parity between desktop and mobile experiences, and the relation of the weight to the performance. Lower scores are better.

Between the original and the optimized version, the CEXi score jumps from 2.38 to 1.03 – a leap that takes it from the measly 22nd percentile of its peers in the IR 500 list of eCommerce sites, all the way to the 88th percentile.

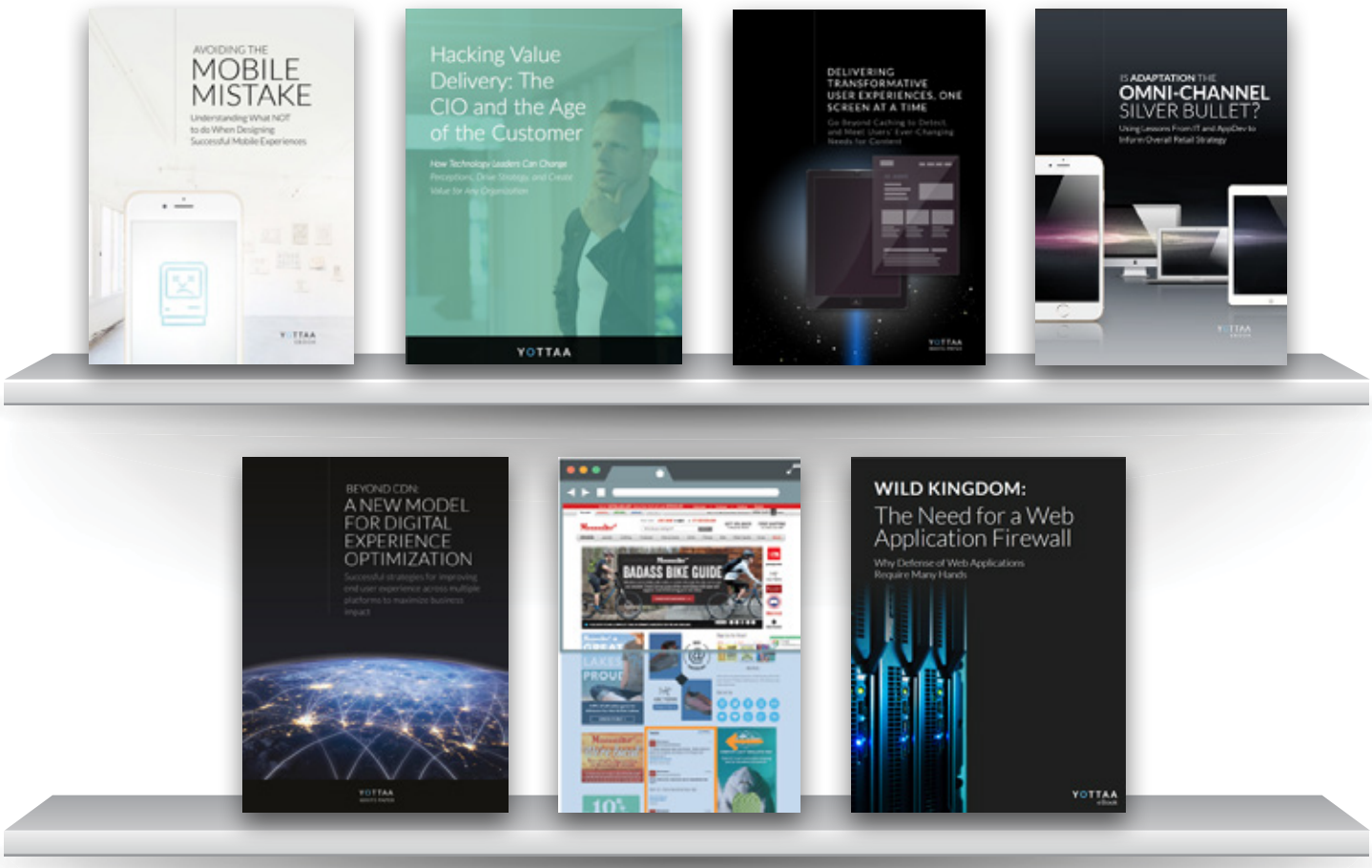
The magic is in content orchestration, the missing piece for optimizing today's complex, dynamic applications. By shifting and re-prioritizing when every individual element loads – including third-parties – we can break the supposed binding relationship between heavy pages and slow speeds.



INTERESTED IN LEARNING MORE ABOUT ORCHESTRATION?

Read our technical whitepaper here:
Orchestrating Web Engagement Requires Context Intelligence

YOU MIGHT ALSO ENJOY READING



About Yottaa

Yottaa is a SaaS solution to manage, optimize and secure digital experience delivery.

Yottaa accelerates online and mobile performance, maximizes end user engagement, and delivers instant, actionable insights to drive business results via an intelligent, automated cloud platform. Our ContextIntelligence™ platform is purpose-built to deliver the power and flexibility required by IT organizations to exceed SLAs for uptime, performance, scalability and security, paired with patented technologies that accelerate the delivery of innovative features and products to improve online and mobile channel execution.

For more information, please visit

WWW.YOTTAA.COM

If you'd like to discuss this paper, or meet with one of our experts to help you expand upon this topic, please feel free to send an email to info@yottaa.com, or contact us toll free in the USA at 1-877-767-0154.

International customers can reach us at +1-617-896-7802. For more details, visit www.yottaa.com