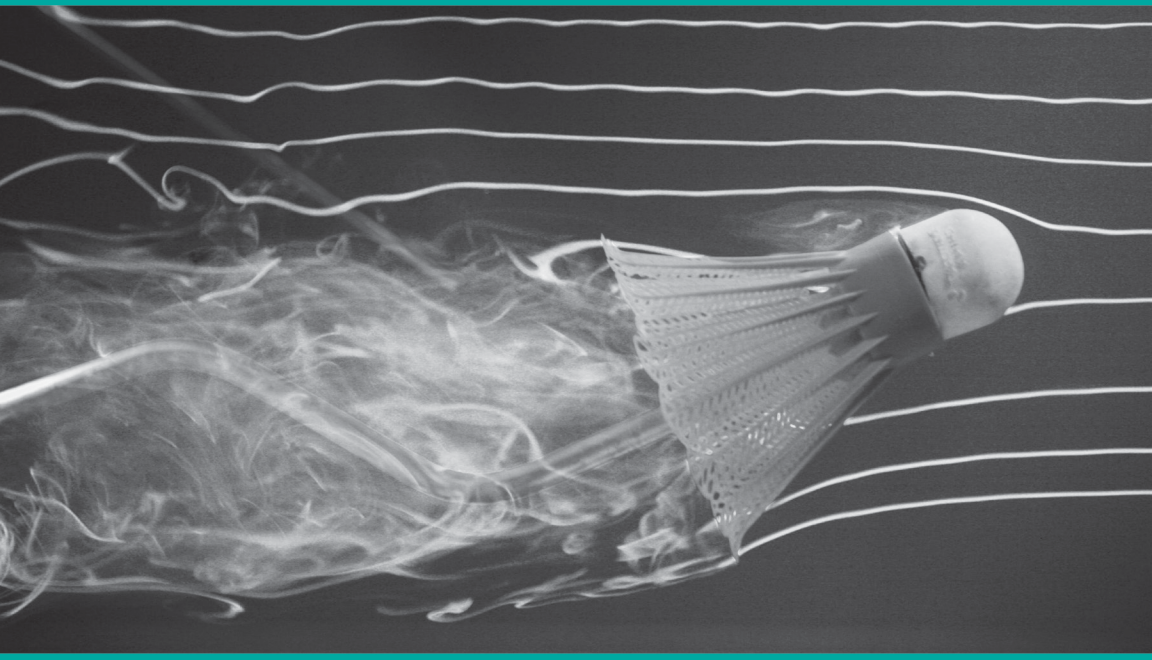
O'REILLY®

Compliments of Hewlett Packard Enterprise

Frontend Optimization Handbook

Ensuring Customer Satisfaction from Your Digital Channels



Larry P. Haig

Accelerating next

Hewlett Packard Enterprise

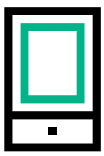
Thrive in the new now: Engineering for the Digital age

Is your application fast enough?

Get your custom **HPE Insights performance report NOW** to learn how your application is performing: www.hpe.com/software/insights.

You will receive a detailed performance report in less than 5 minutes.

Hewlett Packard Enterprise software enables you to deliver amazing applications with speed, quality and scale. **Learn more:**



Mobile testing

Web Performance & load testing

Network Performance

Simulate constrained environment

Frontend Optimization Handbook

Ensuring Customer Satisfaction from Your Digital Channels

Larry P. Haig

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Frontend Optimization Handbook
by Larry P. Haig

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Brian Anderson
Production Editor: Kristen Brown
Copyeditor: Gillian McGarvey
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

May 2017: First Edition

Revision History for the First Edition
2017-05-17: First Release

978-1-491-98500-7

[LSI]

Table of Contents

CHAPTER 1
Preface

"The noblest pleasure is the joy of understanding."
—Leonardo da Vinci, fifteenth century

Welcome to this short handbook on Frontend Optimization (FEO). The vast majority of the response time of digital applications is typically spent at the *frontend* (i.e., on the user's device) rather than in the delivery infrastructure, or in transit between the two. The predominance of mobile users exacerbates the situation, as applications walk a tightrope between ever greater content—advertising, multimedia, etc.—and the relatively limited processing power of many user devices. Although FEO may sound rather arcane, it and its allied discipline of performance monitoring are crucial to the delivery of high performance to end users.

This book covers the importance of FEO from a business point of view but is also a hands-on guide to choosing tools and best practice, as well as a practical approach to understanding and analysis. As such, it is designed to help users get maximum business benefit from FEO. In this book, we will do the following:

- Identify the major opportunities for optimizing performance, and therefore customer satisfaction
- Provide a practical guide to action, including suggested workflow processes
- Support the creation of strategies for ensuring competitive advantage based on efficient digital channels

I wrote this book following a long IT career in major corporations, high-growth entrants, and small consultancies, but my interest in corporate culture extends back over 20 years—my MBA thesis was about my development of the first tool to objectively compare corporate cultures among organizations. Most recently, for more than a decade, I have worked in the FEO field as an analyst and consultant. Throughout that time, I have sought to operate at the interface of IT and business strategy. FEO is a fertile ground for such interaction because, when effectively managed, digital performance is intimately tied to business revenue growth.

Introduction

Most of the material in the following pages first appeared in my end-user monitoring blog. However, this book (hopefully) collates it into a coherent body of information that is of value to new entrants wishing to understand the performance of their web properties, particularly as experienced by visitors to their website(s). I also hope to help readers understand some approaches to detailed analysis of the frontend performance; that is, the components of delivery associated with user devices rather than backend delivery infrastructure.

NOTE

Readers interested in the statistical interpretation of monitoring data should see my short treatment in Chapter 7 of the *The Art of Application Performance Testing, 2nd Edition* by Ian Molyneaux (O'Reilly).

Historically, client-side performance has been a relatively straightforward matter. The principles were known (or at least available, thanks to Steve Souders and others) and the parameters surrounding delivery, though generally limited in modern terms (IE5 / Netscape, dialup connectivity anyone?), were at least reasonably predictable. This doesn't mean that enough people addressed client-side performance (then—or now, for that matter), despite the estimated 80% of delivery time spent on the user machine in those days and almost certainly more today. There is an undoubted association between performance and outcomes, although some of the specifics have elements of urban myth about them. For example, the oft-quoted relationship of 0.1-second deterioration of PC page response with 1% revenue *might* hold true—if you are Amazon.com.

From a monitoring and analysis point of view, synthetic external testing did the job. Much has been written (not least by myself) on the need to apply best practice and to select your tooling appropriately. The advent of real-user monitoring came some 10 years ago—a move that was at first decried, then rapidly embraced by most of the "standalone" external test vendors. The undoubted advantages of real-user monitoring (RUM) in terms of breadth of coverage and granular visibility to multiple user end points—geography, O/S, device, browser—tended for a time to mask the different yet complementary strengths of consistent, repeated performance monitoring at page or individual (e.g., third-party) object level offered by synthetic testing.

Current Challenges

Fast forward to today, though, and the situation demands a variety of approaches to cope with the extreme heterogeneity of delivery conditions. The rise of mobile (as one example, major UK retailer JohnLewis.com quoted that over 60% of digital orders were derived from mobile devices during 2015/16 peak trading) brings many challenges to FEO practice. These include diversity of device types, versions, and browsers, and limiting connectivity conditions.

This situation is compounded by the development of the applications themselves. As far as the Web is concerned, monitoring challenges are introduced by, among other things, full or partial Single Page Applications (SPA), server-push content, and mobile *WebApps*, also known as *Progressive Web Applications* (PWA), driven by service-worker interactions. Mobile applications, whether native or hybrid, present their own analysis challenges, which I will address.

This rich mixture is further complicated by "gravity factor" content drivers from the business—multimedia and other rich content, nonstandard fonts, and more. Increasing amounts of client-side logic, whether as part of SPAs or otherwise, demand focused attention to avoid unacceptable performance in the emergent modern delivery environment.

Emerging Challenges for Web-Based Application Monitoring

- Browser and mobile device diversity
- Single Page Applications
- Progressive web applications
- Multimedia content
- HTTP/2

As if this weren't enough, the emergence of HTTP/2 (finally!) introduces both advantages and antipatterns relative to former best practice. The primitive simplicity of recording page delivery by means of the standard on-load navigation timing point has moved beyond irrelevance to becoming positively misleading, regardless of the type of tool used.

These changes require an increasingly subtle approach combined with a range of tools to ensure that FEO recommendations are both relevant and effective. I will provide some thoughts on effective FEO approaches to derive maximum business benefit in each of these cases. The bottom line is, however, that FEO is more important than ever in ensuring optimal business outcomes from digital channels.

Who This Book Is for

In writing this short book, I envisaged several potential audiences. If you are an experienced technical performance practitioner, this is not the book for you. However—although perhaps not common knowledge—many aspects of performance enhancement are straightforward, and there are a number of use cases where an informed approach to possibilities and processes in this area may promote good practice and enable effective management and business growth.

In short, therefore, this book should be of most use to interested and currently uninformed users in the following categories:

- Senior managers wishing to understand the competitive advantages of high-performance web-based applications, and better challenge existing processes and assumptions in their organizations
- Marketers (or others) responsible for delivering a high-performance digital channel to market, either developed in-house or via external agency
- IT practitioners tasked with optimizing existing web-application performance, in situations where a ground-up rebuild is not a practical option in the short to medium term

The limited space available does not permit an exhaustive treatment, but hopefully it flags some of the cardinal points and acts as a useful introductory, "how to" guide to this important area.

How to Read This Book

It is hoped that readers will gain useful insight from all the content. However, given that is it seeking to address both a line-of-business and technical audience (albeit at a relatively high level), it is possible that some sections will have more relevance than others to your situation.

The material is organized as follows:

Chapter 1 provides a general introduction and emphasizes the goal of performance optimization. As such, it should be relevant to all users.

Chapter 2 outlines the categories of tools available for understanding application performance, and their generic strengths and weaknesses. Most useful to those with a management oversight of application performance or those embarking on monitoring, including delivery to mobile devices.

Chapter 3 is a suggested "how to" process for implementing front-end optimization. It provides a high-level flow process and touches on the key granular areas to address. As such, its primary audience is those wishing to gain a hands-on grasp of FEO in practice, such as someone in an IT role tasked with implementing effective performance optimization but lacking practical experience from elsewhere. It concludes by referencing a source for creating and managing key performance indicators (KPI).

Chapter 4 considers performance from a management/business perspective. It outlines some approaches and potential constraints facing those wishing to take a "root and branch" approach to building a high-performance culture in organizations. It concludes with a brief summary and recommendations about FEO in practice.

Ultimately, it's a short book and should be eminently digestible during a train journey or short flight. I hope that you find it useful.

The Goal of High Performance

The key fact to continually bear in mind regarding FEO is that it is a means to an end, not an end in itself. Many studies have shown the strong association between digital performance and beneficial business outcomes. Many factors influence customer behavior (such as market-set expectation, nature of offer, design, etc.). However, performance is unique in that slow response can trump all other factors. At some point, slow performance inevitably equals lost sales, and user expectations of mobile performance are increasingly convergent with that of the desktop computer.

So, *poor performance equals reduced revenue*. Although some studies have shown that lost traffic does return, either later or via a different channel (e.g., visiting a bricks and mortar store, if it exists, for the brand in question), it is equally likely that this loss will represent some gain to the competition. Figure 1-1 contains RUM-derived data showing the association between transaction abandonment and total page-load time for iPhone and desktop computer users (aggregate US retail sites, one week).

In summary, high-performance sites accomplish the following:

- Minimize transaction abandonment
- Reduce bounce rate from key search engine destination pages
- Build customer satisfaction/loyalty
- Enhance "stickiness" (time on site)
- And thereby maximize competitive advantage.

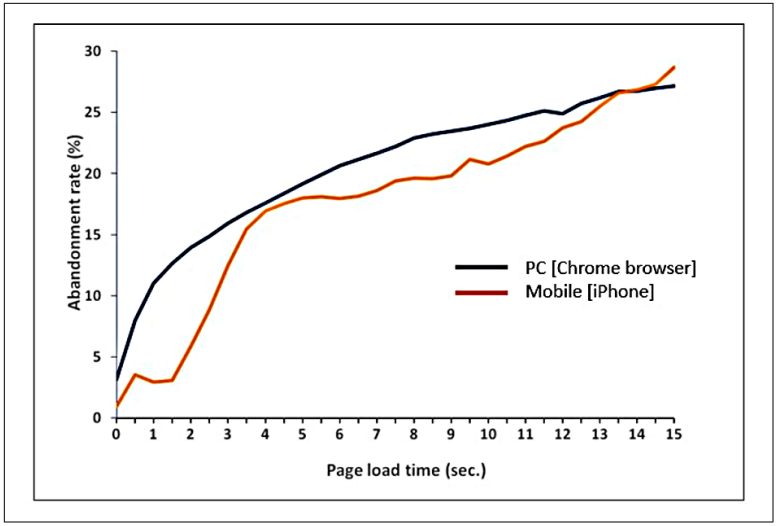


Figure 1-1. Revenue loss: key page response versus transaction abandonment rate (Credit: Gomez Real User Monitoring]

Performance goals are met through *understanding* and *focused intervention*, potentially at all points in the delivery chain from primary infrastructure/content, third-party affiliates, delivery network, or at the client device. This book focuses on the latter, although effective monitoring and analysis will at least reveal weaknesses in the other areas.

NOTE

The key question is: what is the *optimum* performance goal, balancing investment against revenue return?

Although the *high-level* association between application response and revenue has been established since the early days of the commercial internet, this does not particularly assist those responsible for managing performance enhancement for a given site. Between internal analysis and optimization, and adding external enhancements, it is certainly possible to improve the performance of any site. But if this is achieved at ruinous cost, then the net benefit is reduced proportionately.

The ideal is therefore to understand the *tipping point*—the performance at which revenue is optimized but beyond which net returns

are reduced. This figure is unfortunately unique to a given site, although if comparative tests show a gross deficit between your site and its key competitors then it will be best to make a start anyway!

The difficulty in determining the optimal target performance for a given site is that any amount of comparative testing won't tell you the answer—the only people who can do that are your customers. In other words, it is necessary to make an association between actual performance (which varies all the time, of course) and specific revenue-relevant outcomes such as transaction abandonment, page bounce rate, and basket size/total revenue.

Historically, the problem has been that these various metrics are captured by different tools—including RUM performance monitors, behavioral analytics, and order processing systems—and these cannot be correlated exactly due to differences in measurement between them (such as sampling and/or time-to-time aggregation). For example, one (actually, a RUM) tool that purports to display the relationship between performance and revenue is actually using an aggregate average-page-response for *every* page on the site to calculate it.

Trying to "read across" tools is even more fraught with difficulty. This is not to say that one should not try to make the association using the tools at your disposal. You should just be aware that it is very difficult, will only ever be approximate, and will not be worth investing the time unless you have a precise idea of how all the metrics are derived. However, with the advent of holistic Application Performance Management (APM) tools, the goal of effective performance monetization has become much closer—and in some cases, has arrived.

Monetization: The Holy Grail

The ability to report on *transaction* timings at individual user-session level (as opposed to all-user single-page or page-group performance) is particularly useful, although rarely supported. When present, it extends the ability to *monetize performance*; that is, to understand the association between page response to end users and business-relevant metrics such as order size or transaction abandonment.

The creation of such performance-revenue decay curves (for different categories of user), together with an understanding of performance relative to key competitors, enables decision support regarding optimal site performance. This avoids under- or overinvestment in performance.

Modern approaches to monetization use the events-database analytics extensions offered by some APM vendors. *The key factor to establish is that you are measuring a 1:1 relationship between an individual user session, the site response to that session, and the outcome.* The tools that do offer this provide powerful visibility through the ability to ask structured questions of the database, based on linking multiple metrics. This is done using Structured Query Language (SQL)-like scripts. To obtain maximal value, such products should ideally support *relational joins*—making associations between metrics to, for example, compare conversion rates between transaction speed "buckets." It is worth delving into the support (immediate or planned) from a given vendor for the detailed outputs that will underpin business-decision support in this important area.

Figure 1-2 shows an example of monetization output from one of the vendor products.

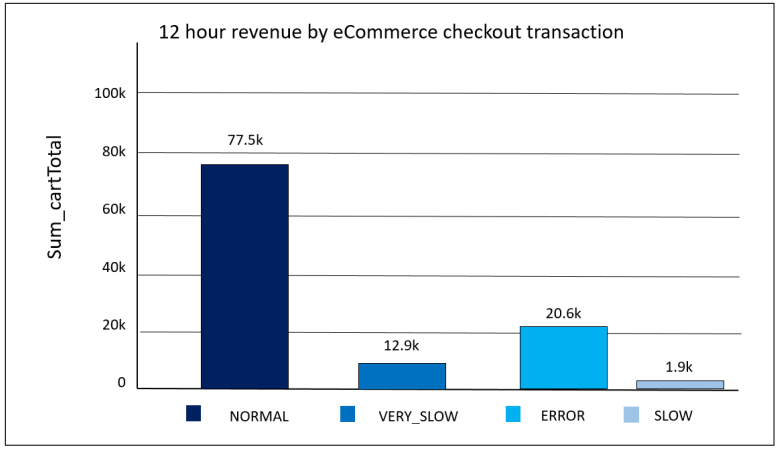


Figure 1-2. Monetization: transaction performance (banded) versus revenue (£) (Credit: AppDynamics]

CHAPTER 2
Tooling

"Measure twice, cut once."
—Carpenter's adage

This chapter considers tool selection. Tools help you understand the current performance of your digital applications—both in ideal conditions and to end users. Such understanding is required on two bases: absolute, and relative to key competitors and other mass market so-called "bellwether" sites such as Facebook, BBC, and CNN—wherever your customers are when they are not on your site.

Tools also let you see how the individual components that make up the overall customer experience of your site are performing—images, multimedia, logic (JavaScript), third-party affiliates, etc. Finally, tools capture the detailed measurements needed to inform core analytics on frontend performance, leading to identification of root-cause issues and ultimately to improved performance.

I will not compare specific vendor offerings, but rather will explain the various generic approaches and their strengths and weaknesses. Success in this field cannot be achieved by a one-size-fits-all approach, no matter what some would have us believe!

Introduction to FEO Tools

I will provide a summary of *available tool types* (see "Relevant Tool Categories for FEO" on page 14) and then a structured FEO process (see Chapter 3). Before doing so, let's start with some high-level considerations. This book assumes an operations-centric rather than

developer-centric approach. Certainly, the most robust approach to ensuring client-side performance efficiency is to bake it in from inception, using established "Performance by Design" principles and cutting edge techniques. However, because in most cases, "I wouldn't have started here" is not exactly a productive recommendation, let's set the scene for approaches to understanding and optimizing the performance of existing web applications.

So, tooling. Any insights gained will originate with the tools used. The choice will depend upon the technical characteristics of the target (e.g., traditional website, Single Page Application, PWA/WebApp, Native Mobile App), and the primary objective of the test phase (covering the spectrum from [ongoing] monitoring to [point] deep-dive analysis).

NOTE

I will use examples of many tools to illustrate points. These do not necessarily represent endorsement of the specific tools. Any decision made should include a broad consideration of your individual needs and circumstances.

Gaining Visibility

The first hurdle is gaining appropriate visibility. However, it must be noted that any tool will produce data, but the key is effective *interpretation* of the results. This is largely a function of *knowledge and control of the test conditions*.

Two Red Herrings

A good place to start in tool selection is to stand back from the data and understand the primary design goal of the tool class. As examples, consider two tools, both widely used, neither of which is appropriate to FEO work even though they are superficially relevant.

Firstly, let's consider behavioral web analytics, such as Google Analytics. Some of these powerful, mass-market products certainly will generate some performance (page response) data. They are primarily designed for understanding and leveraging user behavior, not managing performance. Still, the information that such tools provide can be extremely useful for defining analysis targets both in terms of key transaction flows and specific cases (e.g., top-ranked search engine destination pages with high bounce rates). They are,

however, of no practical use for FEO analysis. This is for several detailed reasons but mainly because the reported performance figures are averaged from a tiny sample of the total traffic, and granular component response data is absent.

Secondly, consider functional/cross-browser test tooling, like Selenium. These are somewhat more niche than behavioral analytics, but they certainly add considerable value to the pre-launch testing of applications, both via device emulation and real devices. All testing originates in a few (often a single) geographic location, thus introducing high and unpredictable latency into the testing. This tooling class is excellent for functional testing, which is what it is designed to do. Different choices are required for effective FEO support.

Key Aspects of FEO Practice

As we will see when considering process, FEO practice in operations essentially consists of two aspects. One is understanding the outturn performance to external end points (usually end users). This is achieved through *monitoring*: obtaining an objective understanding of transaction, page, or page component response from replicate tests in known conditions, or of site visitors over time. Monitoring provides information relative to patterns of response of the target site or application, both absolute and relative to key competitors or other comparators.

The other aspect is *analysis* of the various components delivered to the end-user device. These components fall into three categories: static, dynamic, or logic (JavaScript code). Data for detailed analysis may be obtained as a by-product of monitoring, or from single or multiple point "snapshot" tests. Component analysis will be covered in a subsequent section (see "Component-Level Analysis" on page 51).

What Is External Monitoring?

External monitoring may be defined as any regular measurement of application response time and availability from outside the edge servers of the delivery infrastructure. There are broadly two types of external monitoring approach: *synthetic*, which relies on the regular automated execution of what are effectively functional test scripts, and *passive* (also known as Real User Monitoring, or RUM), which

relies on the capture or recording of visitor traffic relative to various timing points in the web application code.

It is useful to think of FEO as an extension activity supported by specifically targeted monitoring but undertaken separately to "core" production monitoring.

Production monitoring is typically characterized by ongoing recording and trending of defined key performance indicators (KPI); see reference to my detailed treatment of this subject in "Securing Gains with Ongoing Monitoring and KPI Definition" on page 67. These are most effectively used to populate dashboards and balanced scorecards. They provide an extremely useful mechanism for understanding system health and issue resolution, and are often supported by Application Performance Management (APM) tooling.

Relevant Tool Categories for FEO

So what are the relevant categories of frontend test tooling? The following does not seek to provide a blow-by-blow comparison of the multiplicity of competitors in each category; in any case, the best choice for you will be determined by your own specific circumstances. Rather, it is a high-level category guide. As a general rule of thumb, examples of each category will ideally be used to provide a broad insight into end-user performance status and FEO. Modern APM tools increasingly tick many of these boxes, although some of the more arcane (but useful) details are yet to appear—beware the caveat (see "APM Tools and FEO: A Cautionary Note" on page 16)!

As outlined in the next section, tools for monitoring external performance fall into two distinct types: *active* or *passive*. Each are then covered in more detail.

Tooling Introduction

Following is a high-level introduction to the principal generic types of tooling used to understand web application performance and provide preliminary insights for use in subsequent FEO. Tools fall into two main categories, which will be discussed in more detail in subsequent sections. Open source options do exist in each category, although for a variety of technical reasons, these are often best

reserved for the more experienced user—at least when undertaking detailed optimization work.

Firstly, *synthetic monitoring*. This has several subtypes, not all of which may be provided by any given vendor. The principal test variants are:

- Backbone (primary-ISP-based) testing, either from individual Tier 1 (or *quasi* T1) data centers such as Verizon, British Telecom, or Deutsche Telecom or from an Internet Exchange Point (such as the London Internet Exchange [LINX]). The latter provides low-latency multiple tests across a variety of carriers.
- Cloud-based for comparison of relative CDN performance.
- Private peer locations, which can be any specific location where a vendor test agent has been installed. Typically, these are inside a corporate firewall (e.g., sites such as customer service centers or branch offices), although they could include testing from partner organizations, such as an insurance company underwriting application accessed by independent brokers. In theory, such testing could involve Internet of Things (IoT) devices or customer test panels (e.g., VIP users of betting and gaming sites).
- End user testing from test agents deployed to consumer grade devices, connected via so-called "last mile" (e.g., standard domestic or commercial office) connections. Depending upon the technology used, these can vary between "true" end users recruited from the general population in a given country or region, Private Peer testing (see above) or quasi end-user testing from consumer grade devices over artificially modelled connection speeds. WebPageTest provides a good open source example of the latter.

The second type of tooling is *passive*, *visitor*, or *real-user monitoring* (RUM):

- The performance analysis of incoming traffic by reporting of individual or grouped user responses to a variety of timing points in the page delivery process.

- Performance metrics are associated with other user-device related information, such as:
  - — Operating system
  - — Screen resolution
  - — Device type

A subtle variant of RUM is end-user experience monitoring (EUM):

- EUM is essentially RUM (i.e., it's a synonym used by some vendors), but note the distinction between experience in this sense (that is, speed of response) and behavioral-based end-user experience tools and techniques such as click-capture heat maps (see Figure 3-1). The latter are more associated with design-led behavior and support a separate category of tools, although heat-map-type outputs are increasingly being incorporated into RUM tools.

APM Tools and FEO: A Cautionary Note

I will reference *Application Performance Monitoring* (APM) in the context of the various categories considered, although a variety of independent test tools and approaches will typically be used by the FEO practitioner to provide the detailed data required for client-side understanding and tuning.

The provision of such data by an APM, while sometimes possible, if too granular is likely to swamp out the higher-level insights needed for day-to-day steady-state production monitoring. Primarily for this reason, many APM vendors compromise certain aspects of data capture (for example, by data sampling). This is fine for many situations, but effective FEO analysis requires an *absolute understanding of data* and therefore uses more targeted (albeit less comprehensive) tools.

Active (Synthetic) Tooling

The term *active* (sometimes called *synthetic* or *heartbeat* monitoring) is used to describe testing that works by requesting information from the test application from a known, remote situation (data center or end-user device) and timing the response received.

Active Monitoring: Key Considerations

Active (aka synthetic) monitoring involves replicate testing from known external locations. Data captured is essentially based on reporting on the network interactions between the test node and the target site. The principal value of such monitoring lies in the following three areas:

- Understanding the *availability* of the target site.
- Understanding site response/patterns in consistent test conditions; for example, to determine long-term trends, the effect of visitor traffic load, performance in low-traffic periods, or objective comparison with competitor (or other comparator) sites.
- Understanding response/patterns of individual page components. These can be variations in the response of the various elements of the object delivery chain—DNS resolution, initial connection, first byte (i.e., the dwell time between the connection handshake and the start of data transfer over the connection, which is a measure of infrastructure latency), and content delivery time. Alternatively, the objective may be to understand the variation in total response time of a specific element, such as third-party content (useful for Service Level Agreement management).

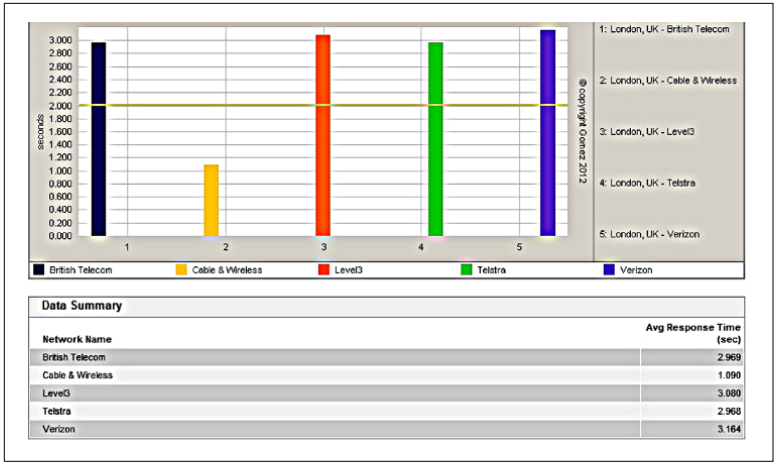
Increasingly, modern APM tools offer synthetic monitoring options. These tend to be useful in the context of the APM (i.e., holistic, ongoing performance understanding), but more limited in terms of control of test conditions and specific granular aspects of FEO point analysis such as Single Point of Failure (SPOF) testing of third-party content. Although it may sound arcane, this is a key distinction for those wishing to really get inside the client performance of their applications.

In brief, the key advantages of synthetic tooling for FEO analysis are these:

- Range of external locations – geography and type
  - Tier 1 ISP/LINX test locations; end-user locations; private peer (i.e., specific known test source)
  - PC and mobile (the latter is becoming increasingly important)
- Control of connection conditions—hardwired versus wireless; connection bandwidth
- Ease and sophistication of transaction scripting—introducing cookies, filtering content, coping with dynamic content (pop-ups, etc.)
- Control of recorded page load end point (see "What Are You Measuring? Defining Page-Load End Points" on page 20), although this also applies to RUM if custom markers are supported by the given tool. As a rule of thumb, the more control the better. However, a good compromise position is to take whatever is on offer from the APM vendor—provided you are clear as to exactly what is being captured—and supplement this with a "full fat" tool that is more analysis-centric (WebPageTest is a popular and open source choice). Beware variable test node environments with this tool if using the public network.

Figure 2-1 is an example of a helpful report that enables comparison of site response between major carriers (hardwired ISPs or public mobile networks). Although significant peerage issues (i.e., problems with the "handover" between networks) are relatively rare, if they do exist, they are:

- Difficult to determine without such control of test conditions
- Have the propensity to affect many customers—in certain cases/markets, 50% or more



Data Summary

Network Name	Avg Response Time (sec)
British Telecom	2.969
Cable & Wireless	1.090
Level3	3.080
Telstra	2.968
Verizon	3.164

Figure 2-1. Synthetic monitoring—ISP peerage report (UK)

End-user synthetic testing. Figure 2-2 is an example from a synthetic test tool. It illustrates the creation of specific (consumer-grade) test peers from participating members of the public. Note the flexibility/control provided in terms of geography and connection speed. Such control is highly advantageous, although it will ultimately be determined by the features of your chosen tool. In the absence of such functionality, you will likely have to fall back on RUM reporting, although bear in mind (as mentioned elsewhere) this is: inferential not absolute, and it will not give you an understanding of availability as it is reliant on visitor traffic.

Name	Create Date	Created by	Capacity	Options
All China HBB	3/25/2013 15:31	Larry Haig	Capacity...	
Australia HBB	8/31/2012 15:26	Larry Haig	Capacity...	
France HBB	8/31/2012 15:32	Larry Haig	Capacity...	
France LBB	10/11/2012 09:55	Larry Haig	Capacity...	
France Saturated	10/11/2012 09:56	Larry Haig	Capacity...	
HBB Ireland	9/20/2012 13:38	Larry Haig	Capacity...	
HBB Netherlands	9/20/2012 20:56	Larry Haig	Capacity...	
India HBB	8/31/2012 15:27	Larry Haig	Capacity...	
Italy HBB	8/31/2012 15:28	Larry Haig	Capacity...	
Larrys Manchester HBB Peers	9/6/2013 11:37	Larry Haig	Capacity...	
LBB Ireland	9/20/2012 16:12	Larry Haig	Capacity...	
Saturated Ireland	9/20/2012 16:13	Larry Haig	Capacity...	
Saudi Arabia HBB	9/27/2012 19:53	Larry Haig	Capacity...	
Saudi Arabia LBB	9/27/2012 19:54	Larry Haig	Capacity...	
Saudi Arabia Saturated	9/27/2012 19:55	Larry Haig	Capacity...	
Singapore HBB	8/31/2012 15:24	Larry Haig	Capacity...	
Spain G4 G5 Larry	11/5/2014 14:04	Larry Haig	Capacity...	
Suisse HBB	9/25/2012 18:55	Larry Haig	Capacity...	

Figure 2-2. Creation of end user test clusters in synthetic testing

What Are You Measuring? Defining Page-Load End Points

Now, an important word on page-load end points. Traditional synthetic tools rely on the page onload navigation marker. It is essential to define an end point more closely based on end user experience (i.e., browser fill time), as this is what is perceived by the user as page response regardless of what is happening "behind the page." With older tools, this needs to be done by introducing a flag to the page. This can either be existing content such as an image appearing at the base of the page (at a given screen resolution), or by introducing such content at the appropriate point. This marker can then be recorded by modification of the test script.

Given the dynamic nature of many sites, attempting to time to a visual component can be a short-lived gambit. Introducing your own marker, if you have access to the code, is a more robust intervention.

It is worth exploring whether a tool will support this feature automatically, thus saving a lot of work.

Figure 2-3 illustrates the manual introduction of a test pixel as a timing end point into the source code of a web application should this not be supported as a standard feature within your chosen tool.

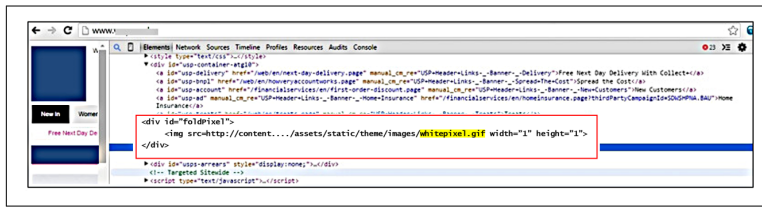
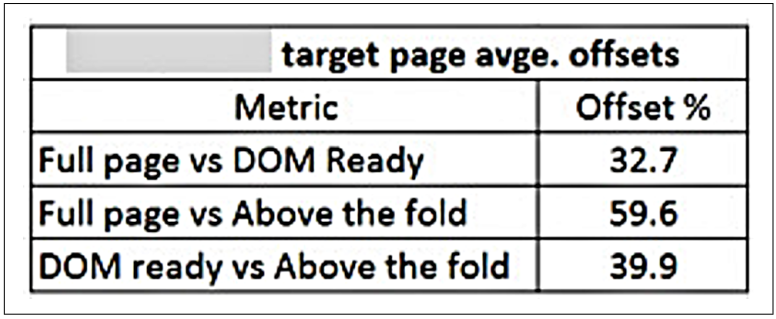


Figure 2-3. "Above the fold" end point; custom insertion of flag image; synthetic testing

Some modern tooling has introduced this as a standard feature. It is likely that competitors will follow suit.

Let me emphasize: using the onload marker will produce results that do not bear any meaningful relationship to end user experience, particularly in sites with high affiliate content loads.

Figure 2-4 illustrates the very high variation in recorded page response depending upon the end point used.



target page avge. offsets	
Metric	Offset %
Full page vs DOM Ready	32.7
Full page vs Above the fold	59.6
DOM ready vs Above the fold	39.9

Figure 2-4. PC page response variation with end point (example)

Modifications of standard testing are required to manage potentially misleading results in specific cases (e.g., server push, Single Page Applications). These are covered in "Emerging Developments" on page 59.

Passive (RUM-based) Monitoring Tools

The previous section considered synthetic (active) monitoring of PC-based sites by examining data from replicate "heartbeat" external tests in known conditions. Now let's consider complimentary monitoring of actual visitor traffic and aspects of mobile device monitoring.

Passive monitoring—also known as Real-User Monitoring (RUM), End User Monitoring (EUM), User Experience Monitoring (UEM)—is based on the performance analysis of actual visitors to a website. This is achieved by manual or (more typically) automatic introduction of small JavaScript components to the web page. These record and return (by means of a beacon) the response values for the page, based on standard W3C navigation metrics—DOM ready time, page onload time, etc. It is worth noting in passing that these are not supported by all browsers, notably older versions of Safari and some others. However, the proportion of user traffic using unsupported versions of non-Safari browsers will probably be fairly negligible today, at least for core international markets.

Figure 2-5 shows a typical RUM dashboard, illustrating near real-time split of visitor traffic by geography, devices, operating system, etc.

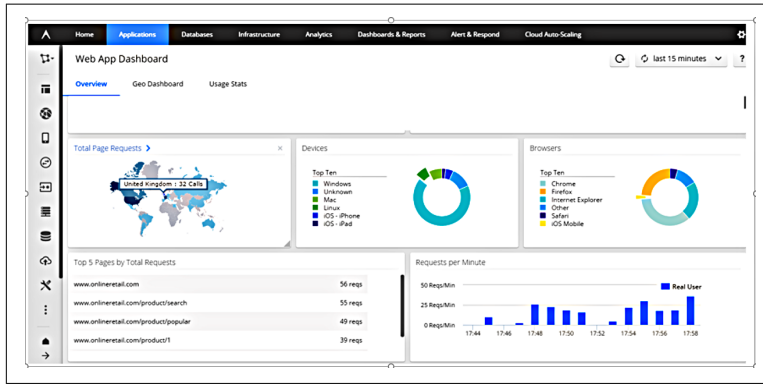


Figure 2-5. A typical RUM dashboard (Credit: AppDynamics)

Modern RUM tooling increasingly captures some information—subject to certain technical limitations outside the scope of this book—at object level as well (or it can be modified to do so). A useful capability, available in some tools, is the ability to introduce *custom end points*. If supported, these can be coordinated with appropriately modified synthetic tests (see "What Are You Measuring? Defining Page-Load End Points" on page 20), providing the ability to read across between active and passive test results. Figure 2-6 illustrates a useful end user monitoring report. The table shows the variation of mobile visitor traffic response for all the target pages on a site.

A further useful capability in some RUM tools is event timing. *Event timing* involves the placing of flags to bracket and record specific user-invoked events (e.g., the invocation of a call to a payment service provider as part of an ecommerce purchase).

Name	Tota	End User Response Time (ms)	DOM Ready Time (ms)	Front End Time (ms)	Resource Fetch	HTML Download	DOM Building	First Byte Time (ms)	Response Available	Server Connection Time (ms)
GALLERY	709,509	7,827	4,294	6,647	3,950	2,712	2,954	950	715	224
PRODUCT	295,704	9,618	5,639	8,935	5,344	3,611	3,592	900	526	360
GALLERY - SEARCH_SORT	119,532	7,481	5,705	6,614	3,903	2,736	2,790	1,017	561	449
HOMEPAGE - HOME PAGE	100,697	8,633	6,468	9,611	4,935	4,739	4,053	1,236	776	457
GALLERY - SEARCH_SUCCESS	60,210	6,799	3,641	5,560	3,352	2,224	2,425	761	587	156
MY ACCOUNT - ACCOUNT LANI	51,333	4,848	2,923	3,972	2,170	1,805	1,995	883	379	503
BASKET - BASKET	49,411	7,986	7,936	9,516	6,596	2,941	3,025	793	379	401
LOGIN - ACCOUNTLANDING	48,752	9,053	9,163	9,155	4,646	4,529	4,596	1,456	360	1,095
ZONES - WOMEN	33,886	6,431	4,740	6,008	3,308	2,755	2,862	641	425	208
MY ACCOUNT - ACCOUNT SUM	32,951	5,086	2,968	3,938	1,692	2,247	2,232	807	648	159
MY ACCOUNT - TRACKORDER	24,643	6,802	3,975	5,383	3,052	2,334	2,411	1,922	1,605	313
CHECKOUT - DELIVERY OPTION	20,536	7,014	4,069	4,434	1,786	2,649	2,668	1,928	602	1,325
GALLERY - SEARCH_REDIRECT	17,090	8,636	6,140	6,540	3,260	3,286	3,558	1,506	554	946
MY ACCOUNT - MAKE PAYMEN	15,151	5,493	3,185	4,234	2,333	1,902	2,008	1,225	882	343
PRODUCT REVIEW - PRODUCT R	14,963	5,284	4,851	3,904	1,661	2,243	2,379	496	343	138
MY ACCOUNT - TRACK PARCELS	12,646	5,606	3,051	4,441	2,586	1,871	2,012	1,125	866	258
CHECKOUT - ADD TO ACCOUN	11,974	5,714	3,563	3,856	1,707	2,152	2,262	1,292	344	948
LOGIN - LOGIN-REGISTER	11,572	6,550	5,237	11,484	8,960	2,540	2,770	988	327	661
CHECKOUT - LOGIN	10,127	9,979	7,632	10,194	4,873	5,335	5,345	1,145	312	833

Figure 2-6. APM End User Monitoring: mobile visitors by usage and response

Performance APIs

I should include a few words on the use of performance-centric APIs. This includes "traditional" navigation flags—DOM Ready, page unload, etc.—that have been around for a few years now, along with more leading-edge developments, such as the `sendBeacon` (already referenced regarding monitoring service worker/push content) property, the `Event.timestamp` property, and others. The only negative to introducing timing APIs in this book is that it moves us across the "dev" spectrum and away from an introduction to day-to-day operations. Failure to exploit them, however, will prove a serious limitation to effective performance practice going forward, so awareness and, if possible, adoption is increasingly important.

Network timing attributes are collected for each page resource. Navigation and resource timers are delivered as standard in most modern browsers for components of "traditional" page downloads. User interaction and more client-centric design (e.g., SPAs), however, require event-based timers. Basic custom timers introduce a timing `mark()` at points within the page/code. Your RUM tooling should be able to support these, and they enable read-across between different tooling (e.g., using visual end points for user experience endpoints and browser fill times in synthetic measurements). Not all RUM products *do* support these, however, so this is an important aspect to understand when making a product purchase decision. Other APIs have been developed to support things like

image rendering and frame timing, which are important if seeking to ensure smooth user experiences.

Browser support cannot be taken for granted, particularly with the newer APIs. It is important to be aware of which browsers support each method, as you will be "blind" with respect to the performance of users with non-supported technologies. In certain cases (e.g., Opera in Russia, or Safari for media-centric user bases), this can introduce serious distortions to results interpretation.

A useful primer for Web Performance Timing APIs, which also contains links to further specialist information in this evolving area, can be found here. Figure 2-7 shows the extremely valuable "Can I Use" reference site. This provides a regularly updated look up of which browsers currently support a particular performance timing API.

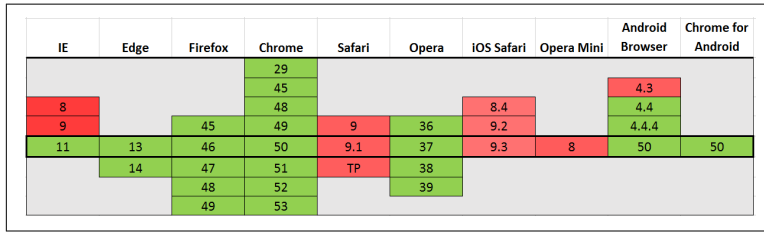


Figure 2-7. Browser support for resource timing API (Credit: Can I Use, May 2016)

Monitoring Mobile Devices

The increasing traffic from mobile device users makes careful consideration of the end user experience a key part of most current FEO efforts. Investigation typically uses a combination of emulation-based (including browser developer tool) analysis, including rules-based screening (e.g., Google's PageSpeedInsights—see "Rules-based screening" on page 44) and real device testing.

The key advantage of testing from real mobile devices as opposed to spoofed user-string/PC-based testing is that the interrelationship between device metrics and application performance can be examined. As discussed in "Active Monitoring: Key Considerations" on page 17, ensuring good, known control conditions is essential. This applies to connectivity (bandwidth, SIM/mobile phone network, or WiFi) and device environment. Both are crucial to effective interpretation of results.

Most cross-device tools are designed for functional (or, in some cases, load) testing rather than performance testing per se. This limits their value. The choices are:

- Limiting investigation to browser-based developer tools.
- Building/running an in-house device lab with access to presentation layer timings and device system metrics (not as trivial an undertaking as it may seem).
- Using a commercial tool—these are thin on the ground, but a few exist.
- Using the real device testing offered by performance vendors. Look before you leap—the devil is in the detail!

Four approaches to understanding the performance of native mobile applications are possible:

- Consider a commercial tool (if you can find one—they may emerge)
- Instrument the application code using a Software Developer Kit (this is the approach adopted by the APM vendors). Typically, these are stronger on end-user visibility rather than control of test conditions or range of device metrics. Crash analytics can be useful, if included (not all provide this).
- Use a Network Packet Capture (PCAP) approach, analyzing the initial download size and ongoing network traffic between the user device and origin. This is the approach taken by the (open source) AT&T ARO tool.
- Build your own in-house device lab with API hooks into device hardware metrics like CPU utilization, memory, battery state, etc. Figure 2-8 is a trace of CPU utilization (user, kernel, and total) by a mobile device during a repeated (three-phase) test transaction.

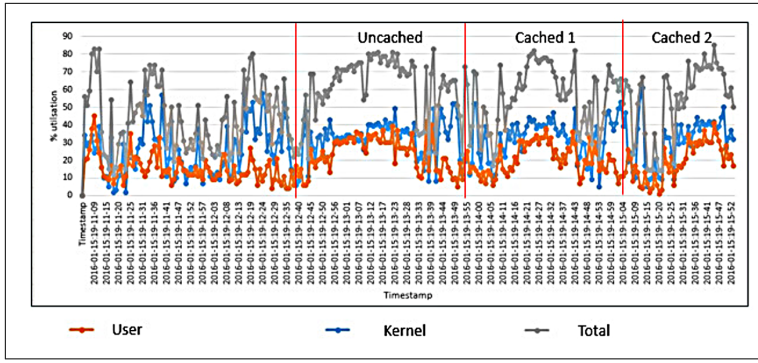


Figure 2-8. CPU utilization trace during test transaction on an Android device

Either way, having defined your control conditions within the constraints of the tool/approach selected, the key aspects are:

- Timeline: understand the interrelationship between the various delivery components, such as JavaScript processing, image handling, etc., and CPU utilization
- System metrics when delivering cached and uncached content. These include:
  - CPU—OS (kernel), user, total
  - Memory—free, total
  - Battery state
  - Signal strength
- Crash analytics
  - Impact of third-party content

Key Categories of Mobile Monitoring

There is a core distinction between *emulation* and *real device* testing.

Emulation testing

Emulation testing has the advantage of convenience and the ability to rapidly test delivery across a wide variety of device types. It also uses a consistent, powerful, PC-based platform. This can be useful depending on the precise nature of the testing. Emulation consists of "spoofing" the browser user string such that the request is presented

to the target site as a mobile device. Given that it is important to replicate (a range of) realistic user conditions to gain an understanding of actual performance in the field, the most useful tools will permit comparison across a variety of connection types and bandwidths, including hardwired, WiFi, and public carrier network. Figure 2-9 shows a 20%-page-size variation at fixed bandwidth for an Android smartphone over a public wireless carrier. Such variation can be much larger on occasion, and with careful analysis the root cause can usually be pinpointed. Bear in mind that such effects can be introduced by specific carriers as well as (more commonly) by the delivery infrastructure. Failure of third-party content is relatively common at high traffic periods (e.g., betting sites during the Grand National steeplechase in the UK, retail sites during Black Friday weekend) due to the inability of a specific third party to cope with the aggregate delivery demands across the sector.

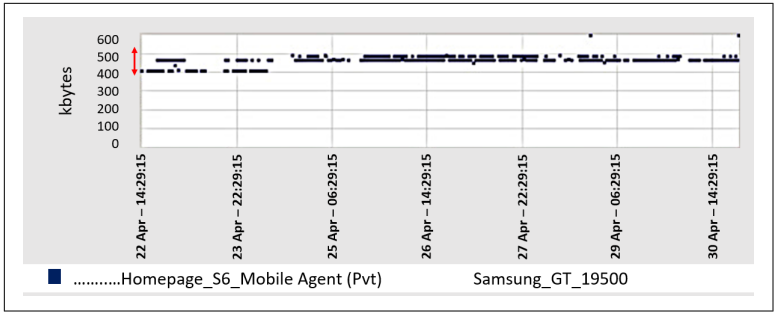


Figure 2-9. Page-size variation at fixed bandwidth (public wireless carrier, Android smartphone)

Many tools (e.g., WebPageTest, browser dev tools, etc.) only offer hardwired connectivity throttled to provide a range on connection speeds. This can be appropriate during deep-dive analysis. It is, however, insufficient for monitoring comparisons. Figure 2-10 illustrates the wide range of "devices" available for emulation-based testing within one market leading browser based developer tool.

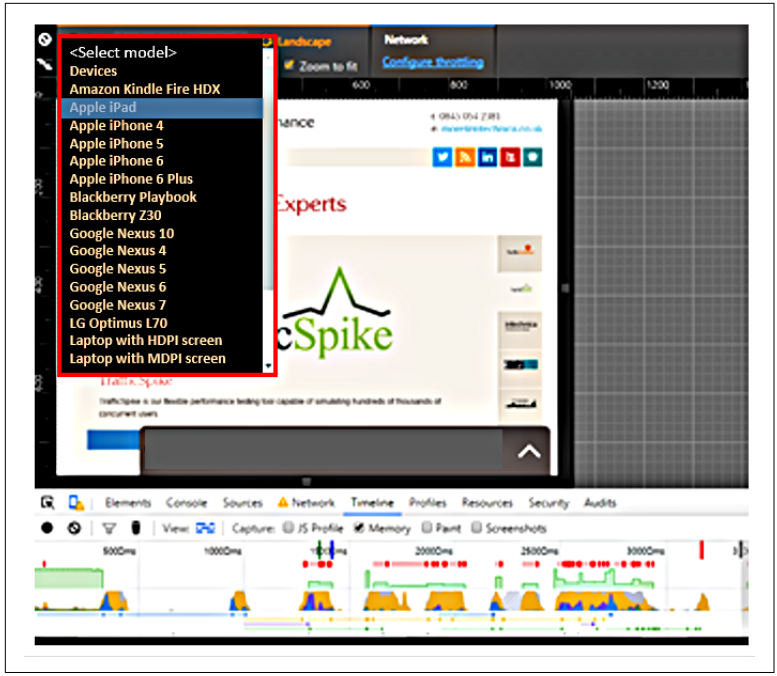


Figure 2-10. Emulation testing: device selection (Chrome Developer Tool)

Real device monitoring

Testing from real mobile devices has several advantages. Access to the graphical user interface for script recording (supported by some tools) enables visual end-point recording. Transactions may be recorded, not only for websites but also native mobile applications. A further advantage of testing from real devices is enhanced control and understanding of the performance influence of device characteristics.

The performance to a given device is likely to be influenced by system constraint. These may be inherent (e.g., processor and memory capacity, operating system version) or dynamic (battery state, memory, and CPU utilization, etc.). In addition, user behavior and environmental factors can have a significant influence—everything from applications running in the background, number of browser tabs open, or even the ambient temperature. Figure 2-11 illustrates device selection within a tool supporting real-device testing.

Figure 2-11. Testing from real device—device selection (Credit: Perfecto)

It's that *control* word again—the more accurate your modeling of test conditions (particularly edge states), the more accurate and relevant your interpretation will become.

Monitoring and Analysis of Native Mobile Applications

Two approaches are possible here. For monitoring/visitor analysis, the most widely used approach (and that adopted by APM tooling) is to provide measurement based on a Software Development Kit (SDK). The application is instrumented by introducing libraries to the code via the SDK. The degree of visibility can usually be extended by introducing multiple timing points, such as for the various user interactions across a logical transaction. Errors are reported, usually together with some crash data.

All the major vendors support both Android and iOS. Choices for other operating systems (such as RIM or Windows Mobile) are much more limited due to their relatively small market share. This limitation should be borne in mind when making decisions about which mobile operating systems to support in creating a native mobile web application. Figure 2-12 shows the use by an APM tool of an SDK code snippet for mobile application instrumentation.

Other tools exist for point analysis of native apps. AT&T's Application Resource Optimizer (ARO) utility is a useful (open source) example. This utility screens applications against best practice in 25 areas based on initial download size and network interactions (PCAP analysis) via a Virtual Private Network (VPN) probe. Figure 2-13 shows the rules-based RAG output offered within this valuable open source tool.

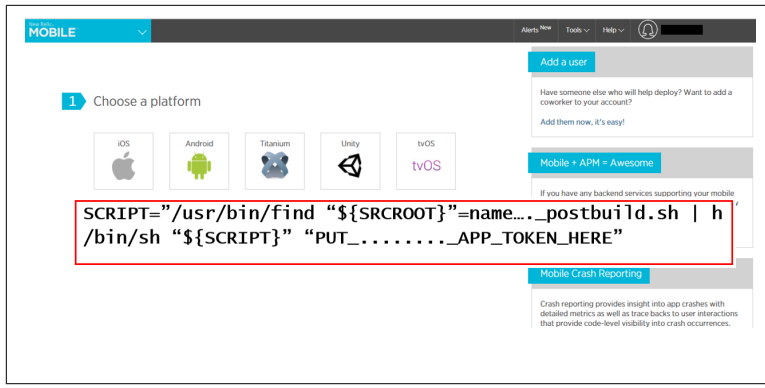


Figure 2-12. Mobile instrumentation using Software Development Kit (Credit: New Relic)

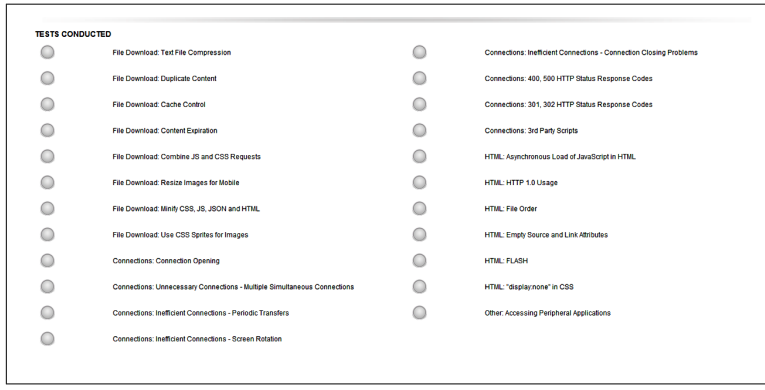


Figure 2-13. Rules-based best practice analysis (25 parameters) for native mobile apps (AT&T ARO)

Most modern APM tools offer both synthetic and passive external monitoring to support end-to-end visibility of user transactions. Although it is possible to integrate "foreign" external monitoring into an APM backend, this is unlikely to repay the effort and maintenance overhead. The key advantage of using the APM vendor's own end-user monitoring is that the data collected is automatically integrated with the core APM tool. The great strength of APM is the ability to provide a holistic view of performance. The various metrics are correlated, supporting a logical drilldown from an end-user transaction to root cause, whether application code or infrastructure.

Be aware, however, that correlation between frontend and backend is unlikely to be at individual user-session level—the RUM data may be, but the backend comparison will usually be to an aggregated time slot. This is probably OK for gross errors, but becomes more limiting the more episodic the issue is. It is important to understand any limitations of the RUM and active test capabilities offered, both to assist in accurate interpretation and to make provisions for supplementary tooling to support deep-dive FEO analytics.

Ultimately, the strength of an APM lies in its ability to monitor over time against defined KPIs and Heath Rules, to understand performance trends and issues as they occur, and to rapidly isolate the root cause of such issues. These are powerful benefits. They do not support detailed client-side analysis against best practice performance-by-design principles. These are best undertaken as a standalone exercise, using independent tools designed specifically with such analysis in mind. The key use case for APM is to support an understanding of "now" in both absolute and relative terms, and to support rapid issue isolation/resolution when problems occur. Figure 2-14 illustrates the ability, offered by most APM tooling, to correlate—to some extent—frontend and backend performance.

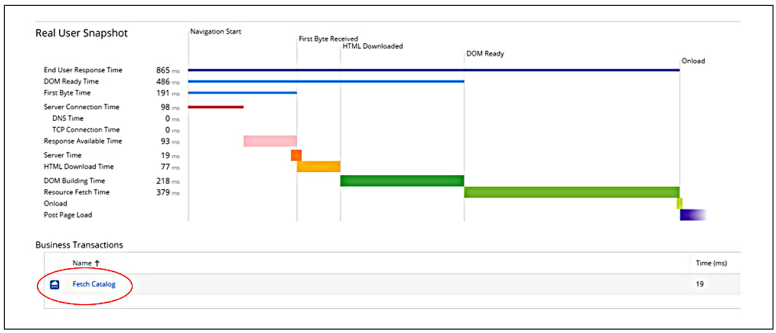


Figure 2-14. Frontend/backend correlation—RUM timings, link (highlighted) to relevant backend data (APM tooling example)

CHAPTER 3
Process

"Plans are of little importance, but planning is essential."
—Winston S. Churchill

As one's knowledge of the complexities and pitfalls around FEO grows, it is easy to begin to feel swamped by the various aspects of effective analysis. This is particularly the case if you have "failed to duck" and have been pitchforked into the responsibility for leading a performance optimization initiative, especially if (as is often the case) such a performance focus has been precipitated by some sort of public failure of the digital brand.

This chapter is an attempt to provide a standard structured approach as a starting point. It may not meet every situation or work for you exactly as presented, but it's a start—and all I can say is that it works for me!

A Structured Process for Frontend Optimization

Having considered the types of tools available to support external monitoring, our FEO "toe dip" continues by outlining a structured process to support understanding and intervention in this important area.

At a high level, a logical FEO process seeks to move progressively from general to specific understanding. The following are the key stages:

1. Target definition and test specification.
2. External performance (response) and patterns.
3. Performance monetization goal: what is the optimal performance/investment that will just meet business goals (see Chapter 1)?
4. FEO effort required: response distribution between frontend, backend, and third-party components.
5. Granular investigation: detailed, component-level, client-side analysis.
6. Securing the gains, including ongoing monitoring and KPI definition (best case external plus end user).

This list is effectively technology-independent in that it could apply to anything being analyzed, although the tools used will differ between specific cases.

For completeness, I end this chapter (see "Emerging Developments" on page 59) with a brief consideration of several more recent developments in internet application delivery or those (such as bots) that affect performance while not being FEO-related, per se. IT is always evolving; the challenge to tool vendors and analysts alike is to evolve approaches to understanding and mitigating negative effects.

Defining Testing Targets

Before embarking on any actual analysis, it is worth pausing to define the targets for the testing. Such target definition is likely to collate information from multiple sources, including:

- Knowledge of the key user touchpoints, such as landing pages, product category pages, shopping basket, etc. In thinking about this, a useful guide is to follow the money; in other words, track key revenue generating paths/activities.
- Information derived from web analytics. This useful source will identify key transaction flows (any unusual patterns from theoretical expectations may reflect design or performance issues). Areas of the site associated with unexplained negative behaviors should be included. Examples include transaction steps associated with high abandonment, high discrepancies between

destination page traffic (e.g., search-engine-derived), and high bounce rate.

- If available, user click-pattern heat maps can also be a useful supplement (see Figure 3-1).

Figure 3-1. Visitor interaction click-pattern heat map (Crazyegg.com)

- Hearsay knowledge: subjective input from multiple sources, such as internal users, friends and family, customer services, CEO's golfing partners, etc.
- Other visitor-based analysis (e.g., Real User Monitoring); in particular, key markets, devices, operating systems, screen resolutions, and connectivity distributions. The latter is particularly useful if supported by your APM tool.
- Marketing/line-of-business input: who are the key competitors (by market), and can we learn anything from digital revenue data?

This will lead to a definition of the test parameters. The more core and edge-case conditions tested, the better the overall understanding, but as with everything, these will be limited in practice by time and money.

Raw visitor-derived data (i.e., from the field, not a usability lab) is (obviously) the outcome of *actual experience* rather than objective, controlled test conditions. For example, a low proportion of low-specification mobile devices may just be a reflection of user demographics or they may reflect user satisfaction issues. This is where validation of RUM inferences using synthetic testing in controlled conditions is particularly useful.

Reference List of Test Parameters

Having considered input from the various sources listed (and potentially others, like direction given to users of closed systems such as business partner ordering applications—e.g., "This application has been designed to function best using x browser, y operating system, z device", etc.), a test matrix should be constructed. Table 3-1 is a typical example that lists the parameters for testing.

Table 3-1. Reference list of test parameters (example)

Parameter	Value
PC Browser(s), versions	Edge, Chrome 48, IE 10,11 FF 34, 44
Screen resolution(s)	1366x768
Mobile device (web)	Samsung Galaxy 5 and 6, IPhone 6, etc.
Mobile device, O/S (Mobile Apps)	Samsung Galaxy 6 Android
Mobile App details, source	Xyz.apk, Google Play
Connection bandwidth range (hardwired and wireless)	0.25–2.0 Mbps
Target ISPs and wireless carriers (by market)	[UK, hardwired] BT, Telstra; [UK wireless] EE, Vodafone
Key geography(ies)	UK; S Spain (Malaga), Hong Kong
Competitor sites (target details), by market	
Other factors (e.g., user details associated with complaints)	

Evidence from Monitoring

Armed with the target specifications for testing, it is useful to begin by *monitoring* the outturn performance of the site/application. Such monitoring should represent a broad range of user conditions and identify patterns of response behavior across a relatively extended

period of use (perhaps two weeks for "business as usual" data, together with peak load periods as a comparison).

High-level patterns

Applications do not exhibit a "straight line" response. Typically, performance (as measured from optimal—not end user—locations) will show a more or less regular sine wave of average response time, reflecting regular variation in traffic across a daily or weekly business cycle. That is OK, but it needs to be borne in mind that the actual situation is far more complex, given the diversity of user conditions in terms of, for example, geography, ISP, network connectivity (WiFi, public carrier, hardwired), mobile device, browser, and more. Other high-level patterns are possible (for example, the progressive reduction in performance followed by sudden reversion to normality, as seen in memory-leak situations), but if the data is tightly controlled, it is fairly rare to find random variation in performance. It can happen, though, and when it does, it tends to consume much time in isolation and resolution.

For these reasons, it is essential to consider three key aspects when seeking to understand performance for FEO (as opposed to business-as-usual production monitoring purposes). These are:

- To understand and *maintain absolute control* of test conditions (for active testing) or the nature of the user(s) (for passive or RUM testing).
- To *avoid data aggregation*—sampling is of little use, and averages are useless. Examine scattergrams of raw data wherever possible.
- Use *selective filtering* for comparison—for example, by comparing response according to mobile connection bandwidth, or between high and low traffic periods.

These approaches should enable the analyst to determine the distribution of response behavior, examine the relationship between core response and outliers, and drill down to understand the broad causation of low performance behavior. Such insights form the foundation of subsequent systematic FEO analysis. Without them, it is difficult to understand the business impact of any issue, and to direct effort accordingly.

The special case of applications based in shared data centers should be mentioned. In situations where multiple (often templated) sites share some aspects of a common infrastructure, it is possible for high traffic to a few sites to cause depletion of delivery resources to others. An example of this situation during Black Friday peak trading is shown later in Figure 3-6.

FEO effort required

As poor performance can be the result of many primary causes (and, indeed, the interaction between more than one), it is important to determine the extent to which client-side issues are involved before committing resolution effort that may be best spent elsewhere. This can be determined by examining the overall time distribution involved in application performance, split across three high-level areas: delivery infrastructure, network, and client.

A number of tools will assist here. From a Freemium perspective, the old dynaTrace AJAX edition utility was particularly effective but, sadly, is now discontinued (and it was also only a PC browser tool). Many of the modern APM tools will fill the void, albeit with differing levels of granularity and elegance.

Such preliminary monitoring enables us to understand what we are up against from a frontend optimization perspective, and ultimately whether we are looking at fine-tuning or wholesale interventions. Use of APM tooling can be particularly useful at this initial stage, both in understanding the relative proportion of delivery time associated with client-side versus backend processing (see Figure 3-2), and in isolating and excluding any issues associated with delivery infrastructure or third-party web-services calls. However, as the external monitoring extensions to APM tools are still evolving in functionality (particularly in relation to synthetic testing), additional tools will probably be preferred for more detailed FEO investigation, due to the better control of test conditions and/or granular analysis offered by more mature and differently positioned products.

In capturing this baseline data, it is important to compare both consistent and end user conditions. Ideally, both visitor-based (RUM) and synthetic-based data should be used. This will give useful information regarding the performance of all components in all traffic conditions.

If at all possible, and as mentioned above (see "What Are You Measuring? Defining Page-Load End Points" on page 20), common above-the-fold (perceived render time) endpoints should be introduced as custom markers. This will greatly assist in reading across between the test types. Such modifications provide a more realistic understanding of actual end user response, although they would be somewhat cumbersome to implement across a wide range of screen resolutions. As noted earlier, some modern APM tools will automatically inject such flags.

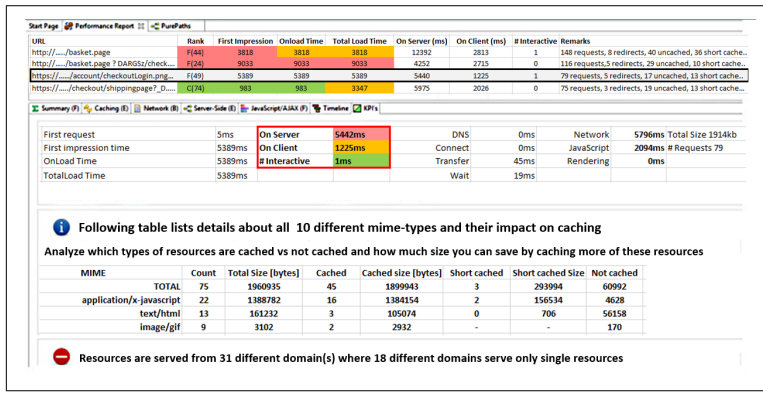


Figure 3-2. Client-side versus server-side processing time (dynaTrace [AJAX Edition])

Subpage-Level Performance

Depending on the detailed characteristics of the target sites, it is often useful to run several comparative monitor tests. Some specific cases (such as Single Page Applications and server push content) are covered in "Emerging Developments" on page 59.

It is often useful to understand the impact of particular—for example, third-party affiliate—components on overall response. This can be achieved in a number of ways. Two of the most straightforward are to test for a SPOF (Single Point of Failure)—i.e., the effect of failure of particular (often third-party) site content—or to remove content altogether. Techniques for achieving this will depend on the tool being used. See Rick Viscomi et al's *Using WebPageTest* (O'Reilly) for details in relation to that tool. The same intervention can be made in most synthetic tools (with more or less elegance) using the relevant scripting language/utility.

Selective filtering of content can also be used to examine the effect of particular calls on aggregate delivery metrics such as DNS resolution or content delivery times.

This screening provides background intelligence prior to beginning detailed analysis of frontend components. Before covering these, now is probably a good time to introduce a standard monitoring matrix that I typically use to provide a structured approach to preliminary screening of external performance. The results are used to inform and direct detailed granular analysis, an overview of which is covered on subsequent pages.

Model Checklist for FEO Preliminary Screening

This 13-point checklist provides a picture of the revenue-relevant behavior of the site/application. This is not exhaustive—it should be led by the findings from case to case—but it supports targeting for the more granular, static component-level analysis that provides the root cause and business-justification basis for specific remediation interventions (see "Granular Analysis" on page 42).

- Dynamic performance: page onload and perceived render (above the fold)
  - 24x7 availability and response patterns: synthetic ISP (by market)
  - 24x7 availability and response patterns: end user by market (synthetic and RUM)
    - Target browser/screen resolution and device
      - Any cross browser/device discrepancies?
    - Defined connectivity: hardwired and public wireless carrier
  - Page response distribution
    - Histogram of response ranges
  - Median response and distribution (median absolute deviation)
    - Weekly business hours
    - Day versus night (variation with traffic)
    - Cached versus uncached
    - By key market/user category

    - Performance monetization (tool-dependent, examples)
        - Page/transaction response versus shopping cart conversion
        - Page/transaction response versus abandonment
        - Page/transaction response versus mean basket size
        - Page/transaction response versus digital revenue (defined time period)
        - Page response versus bounce or exit rate
    - Competitive comparison: direct and mass market sites
        - Page and revenue bearing transaction (e.g., search and add-to-basket)
    - Limiting bandwidth tests
        - Response to progressively reducing connectivity conditions
            - WiFi and public carrier
    - Transaction step comparison
        - Where are the slowest steps (and why—e.g., database lookup)
    - Payload analysis
        - Page download size patterns
    - Affiliate load: third-party effects
        - Filter and SPOF testing
    - Real-device mobile testing
    - Component splits/patterns
        - DNS/SSL resolution
        - Connectivity
        - First-byte time delivery (infrastructure latency)
        - Content delivery
    - CDN performance assurance
        - Origin versus local cache comparison
- Detailed static component analysis

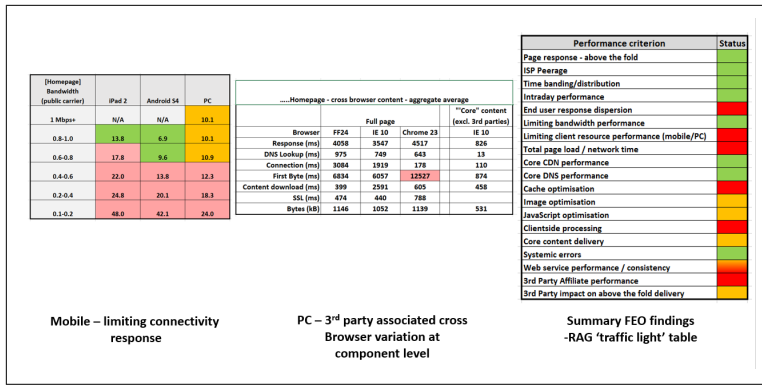


Figure 3-3. Examples of summary outputs from screening [L-R]: Response to limiting connectivity bandwidth (Mobile carriers); Third-party associated cross-browser response variation (component level); RAG based findings summary

Granular Analysis

So where have we got to? Those who have read all the content to this point will have:

- An understanding of the potential benefits of frontend optimization
- An appreciation of the various types of tooling available
- A suggested how-to process for effective results in this area

Having understood the external performance characteristics of the application, in both "clean room" and, more particularly, in a variety of end-user monitoring conditions, we now approach the core of frontend optimization. Monitoring will give a variety of whats, but only detailed granular analysis will provide the whys necessary for effective intervention.

The initial monitoring activity should have provided a good understanding of how your site/application performs across a range of demand conditions. In addition, regardless of the absolute speed of response, comparison with the performance of competitor and other sites should indicate how visitor expectations are being met, and the initial goals for improvement.

Know Your Constraints

Before engaging in hand-to-hand combat with the various client-side components of your site, it is worth taking time to ensure that whoever is charged with the analysis knows the site in detail. How is it put together? What are the key constraints—business model, regulation, third-party inclusions, legacy components—it's a long list. While being prepared to challenge assumptions, it is good to know what the givens are and what is amenable to modification. This provides a good basis for detailed analysis. The team at Intechnica (where I worked for several years) typically adopt a structured approach, as outlined in the next sidebar, bearing in mind that the focus of investigation will differ depending on what is found during the early stages.

As this material is aimed at the "intelligent but uninformed" rather than leading-edge experts, it is also worth ensuring that you are aware of the core principles. These are well covered in many published texts, although things move quickly, so the older the book, the more caution is required. A short suggested reading list is provided in Appendix B.

In summary, a logical standard flow could look rather like this sidebar.

Analysis Phase: Logical Flow Process

The bullet points below provide a suggested structured approach to detailed analysis. Some variation may be preferred, but it is important that information/focus builds progressively from one stage to the next.

- Rules based screening
- Monitoring anomaly investigation
- Component-level analysis
- Network-based investigation
- Recommendations and ongoing testing

This applies to all investigations, although tooling will differ depending on the nature of the target application. I take a similar approach to all PC-based applications. Analysis of delivery to

mobile devices—whether web, webapp, or native mobile applications—benefit from some additional approaches, and these are also summarized in "Monitoring Mobile Devices" on page 24.

Flow Process: Granular Analysis

We have now determined testing targets and understand at a high level the dynamic "outturn" performance of the application—both in terms of individual pages (if present) and specific functions, but also key business transactions. It is now necessary to move from the what to the why: the underlying causation of poor performance.

Clearly, root causes can lie at many doors—backend infrastructure, network performance, third-party interactions, and others—but this book concerns itself primarily with frontend (that is, user-device-based) issues. As such, the following is a suggested sequential process for rapid screening of FEO causation:

- Preliminary rules-based screening
- Anomaly detection/investigation
- Network-level investigation
- Component-level analysis
  - — Client-side logic (JavaScript)
  - — Images/multimedia content
  - — Other aspects
    - — Affiliates
    - — (CDN)[1]
    - — (Backend interactions)

It may well be that isolation of a specific issue involves crossing many of these categories, and a sound knowledge of the architecture of the application will always stand the FEO analyst in good stead.

Rules-based screening

Flippantly, traditional rules-based tools have the advantage of speed, and the disadvantage of everything else! Not quite true, of course,

1 Options in parentheses are included because they add breadth and context to many investigations—not "core" FEO analysis, but useful allied investigations.

but it is necessary to interpret results with caution for a number of reasons, including:

- Developments of technology and associated best practice (e.g., adoption of HTTP2 makes image spriting—formerly a standard recommendation—a potential antipattern)
- Limitations of practical interpretation/priority (e.g., rules based on the percentage gains from compression can flag changes that are small in absolute terms)
- Just plain wrong (e.g., rules that interpret CDN usage as "not using *our* CDN")

Perhaps for a combination of these reasons, the number of free, rules-based screening tools is rapidly diminishing: YSlow and (the excellent) SmushIt image optimization and dynaTrace AJAX Edition tools have all been deprecated over the last year or two. Page Speed Insights from Google is a best-in-class survivor. This is incorporated within several other tools. It provides speed and usability recommendations for both mobile and PC.

So the message is that rules-based screening is a good method for rapidly getting an overall picture of areas for site optimization, but a) use recent tools and b) interpret judiciously. Figure 3-4 illustrates a Chrome-based example.

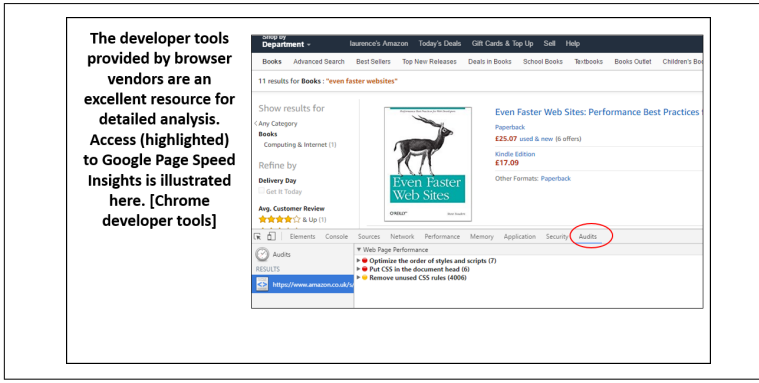


Figure 3-4. Browser-based developer tool

Rules-based screening should provide an insight into the key areas for attention. This is particularly valuable in time-intensive screening of multiple components (e.g., cache settings). Figure 3-5 provides an example of a comprehensive, clear, RAG "traffic light"

output. Note the score disparity between Server (A, left) and front-end components (F, right).

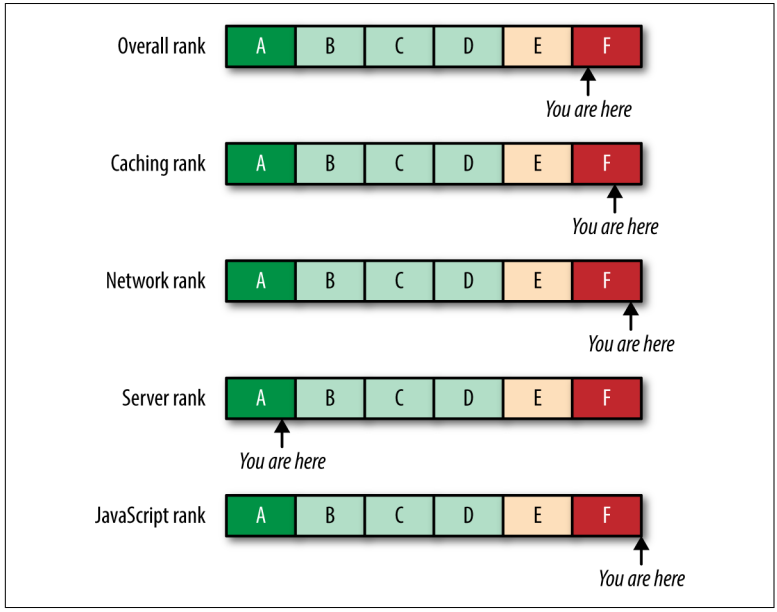


Figure 3-5. Rules-based tool (dynaTrace AJAX Edition)

Monitoring anomaly investigation: slow versus fast versus median

A logical next step is to investigate the underlying root cause of anomalies highlighted in the preliminary monitoring phase. Average traces are useless (for all except possibly long-term trend identification), so it is necessary to identify outliers and other anomalies based on *scattergrams* of raw data. Seek to associate underlying causes. Prior to detailed drilldown, consider possible high-level effects.

Common among these are *traffic* (compare with data from RUM or web analytics), poor resilience to mobile *bandwidth limitations*, and delivery *infrastructure resource* impact from background batch jobs or crossover effects in multitenant providers (see Figure 3-6).

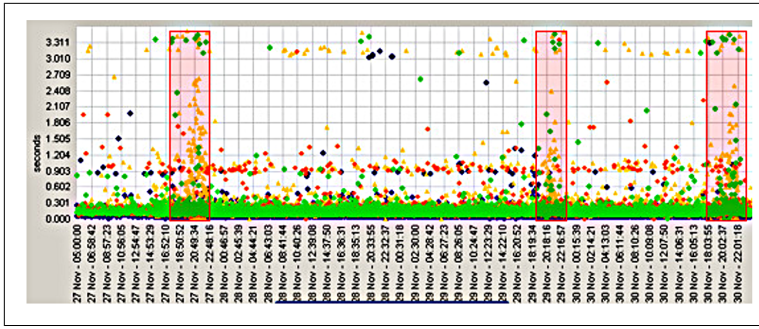


Figure 3-6. Multitenant platform effects: base HTML object response during Black Friday peak trade weekend 2015 (reference site in blue)

Figure 3-6 shows a scattergram of "base HTML object" response. This is a homepage-derived metric that excludes potential sources of comparative confusion between sites; it is the raw response of the site in question. The various sites supported by the platform are represented by different colored dots. As may be expected, the baseline response of the individual sites differs. Most of the time, the response of a given site falls into a more or less defined band of response. In the high-traffic period illustrated, sites will occasionally show a delayed response due to inability to cope with user volume at a point in time. The blue dots (representing a non-multitenant site) show good examples of this—the response is mostly good, but there are periods, such as the example in the middle of the chart, where this is not the case.

With the other sites (i.e., those on the shared hosting platform), the situation is more interesting. Again, baseline responses vary, but note the periods (highlighted in red) in which platform stress has led to multiple sites being affected. Typically, this is because some core shared-platform resource has become depleted due to the overall traffic volume. This causes slower responses to all sites on the platform, whether or not they are individually under stress.

The amount of detail available will obviously depend upon the tooling used for the initial monitoring, although recurrent effects, if identified, should enable focused repeat testing with other, more analytics-focused products such as WebPageTest.

A few notes may be helpful:

- Statistical analysis of individual components is powerful. I suggest that you compare *maximum*, *minimum*, and *dispersion* of individual components (DNS time, connect time, etc.) from median and outlier responses. Progressively remove specific content (e.g., third-party tags) and compare the effect.

Some examples:

- Understand *intraday stress* from traffic patterns on the site (typically using behavioral analytics tools or RUM; see Figure 3-7).

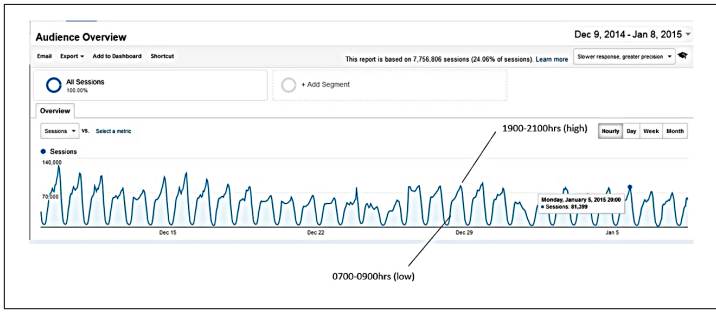


Figure 3-7. Daily traffic patterns to major UK eCommerce site (Google Analytics)

- Use *time-banding comparison* to understand the "pinch points" associated with poor performance at component level (see Figure 3-8).

Page	Time	DNS		Connection		1st Byte		Content Download		SSL Encryption		Total Bytes		End-to-end time	
S12_Women Category page	08:20:29	6.308		6.75		24.234		7.102		0.841		1004837		9.015	
S12_Women Category page	08:40:20	2.627		4.665		21.219		9.335		0.901		1034164		8.878	
S12_Women Category page	08:45:21	4.846		5.8		15.132		4.93		1.081		1033210		6.759	
S12_Women Category page	08:50:18	2.534		5.35		17.632		4.274		0.731		1034079		6.504	
S12_Women Category page	07:30:33	3.922		23.578		34.407		35.727		3.385		1001152		20.862	
S12_Women Category page	07:35:35	3.744		3.962		14.633		2.457		0.607		1003810		8.762	
S12_Women Category page	08:05:24	2.311		5.536		20.418		11.583		1.092		1003904		9.213	
S12_Women Category page	08:15:23	3.549		7.687		14.413		5.207		1.856		88316		6.592	
S12_Women Category page	08:30:28	3.262		7.039		17.854		3.408		1.119		1002710		6.876	
S12_Women Category page	07:00:27	4.489		9.943		21.427		1.518		0.886		1001171		6.733	
S12_Women Category page	07:05:29	5.269		13.81		37.518		13.862		2.816		944472		13.897	
S12_Women Category page	07:35:12	2.793		5.321		17.922		4.319		0.849		1001105		6.959	
S12_Women Category page	07:40:10	5.469		4.177		33.431		13.139		0.583		994816		14.017	
S12_Women Category page	08:00:22	6.474		4.869		14.084		1.832		0.794		1001287		6.325	
S12_Women Category page	08:05:25	1.117		3.586		12.392		6.797		0.672		932242		8.329	
S12_Women Category page	08:15:22	3.236		6.397		16.776		2.744		0.675		963902		6.256	
S12_Women Category page	08:45:41	6.926		6.798		15.308		1.555		1.357		993989		6.553	
S12_Women Category page															
Daily Traffic [7-9am / 7-9pm]		*Low*	*Peak*	*Low*	*Peak*	*Low*	*Peak*	*Low*	*Peak*	*Low*	*Peak*	*Low*	*Peak*	*Low*	*Peak*
	Median	*3.24*	*3.34*	*5.35*	*5.40*	*16.41*	*17.18*	*5.21*	*5.13*	*0.85*	*0.95*	*1,033,210*	*1,002,784*	*6.76*	*7.14*
	Max	*8.06*	*41.95*	*23.58*	*31.47*	*71.95*	*68.33*	*35.73*	*60.05*	*3.39*	*5.94*	*2,264,304*	*1,096,439*	*30.99*	*26.29*
	Min	*1.12*	*0.86*	*3.19*	*2.67*	*10.23*	*7.98*	*1.36*	*1.29*	*0.33*	*0.37*	*88,316*	*157,691*	*3.01*	*3.50*
	MAD	*1.17*	*1.60*	*1.39*	*1.13*	*3.00*	*4.78*	*3.38*	*2.59*	*0.23*	*0.30*			*1.26*	*2.67*
	MADe	*1.73*	*2.37*	*2.06*	*1.67*	*4.44*	*7.08*	*5.00*	*3.84*	*0.34*	*0.44*			*1.87*	*3.96*
	Std Dev	*1.97*	*7.49*	*4.03*	*5.59*	*11.86*	*11.90*	*8.18*	*10.71*	*0.72*	*1.11*			*5.38*	*5.60*
	Skew (SD-MADe)	*0.24*	*5.12*	*1.97*	*3.92*	*7.42*	*4.82*	*3.18*	*6.87*	*0.38*	*0.67*			*3.51*	*1.64*

Figure 3-8. Intraday analysis: peak versus low traffic

- Consider the effect of *third-party affiliate* content (see Figure 3-9).

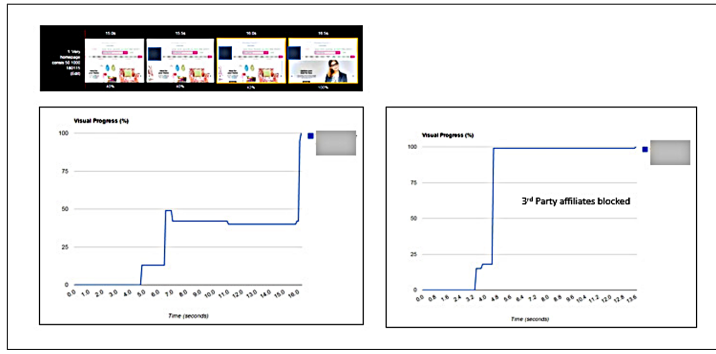


Figure 3-9. Visual progress charts with and without third-party affiliates (WebPageTest.org)

Beware distortion, particularly if page-load end points have been insufficiently well defined (as previously discussed). Waterfall charts should always be inspected to detect gotchas such as below-the-fold asynchronous content or server push connections. Caution needs to be exercised in interpretation of short responses as well as long. Compare payloads; these are often impacted by variable implementation of server-side compression, or content failure.

APM Baseline data may be useful here—although baseline management deserves a book to itself!

Network-Based Investigation

Synthetic test tooling essentially operates by analyzing network interactions between delivery infrastructure and user client (with varying degrees of sophistication). Much of this information is derived from network packet capture (PCAP) data. This is usually represented graphically using a so-called waterfall chart. A *waterfall chart* is simply a graphical representation of a sequential process. Individual line items (y axis) represent different components in the order called by the browser, and the x axis is the timeline. The duration of network interaction of a particular component is shown by a bar. Such bars are often color-coded to illustrate the type of compo-

nent displayed (images, JavaScript, etc.) A waterfall chart is an extremely useful mechanism for rapid understanding of delivery constraints.

Earlier, Figure 2-14 showed a simple (APM-based) example. Interpretation of waterfall charts (as used within the WebPageTest tool) is well covered in Rick Viscomi et al's *Using WebPageTest* (see Appendix B). Such charts can provide much useful information in addition to the parsing of individual object delivery (i.e., partitioning between DNS lookup, connection, first byte, and content delivery times).

Such aspects as blocking behavior, asynchronous/synchronous delivery, and the absence of persistent connections are clearly shown. Side-by-side comparison of waterfalls with different page response time outcomes is a useful technique supported by some tooling. Figure 3-10 shows an example of (third-party) JavaScript blocking activity (arrowed), in this case due to script interaction with 3,000+ DOM nodes.

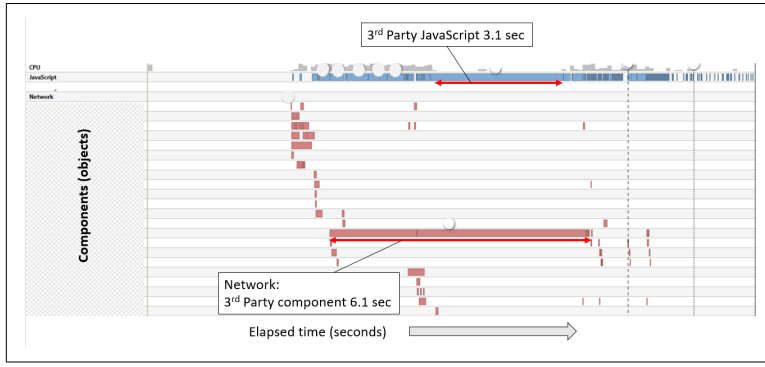


Figure 3-10. Component waterfall chart showing JavaScript blocking (Chrome Developer Tools)

In certain cases, it might be useful to examine traces from "full-fat" network tracing tools (such as Gomez Transaction Trace Analyzer or WireShark). Figure 3-11 illustrates the use of Cloudshark to investigate/confirm server response latency by identifying any delay between network SYNchronise and ACKnowledge responses.

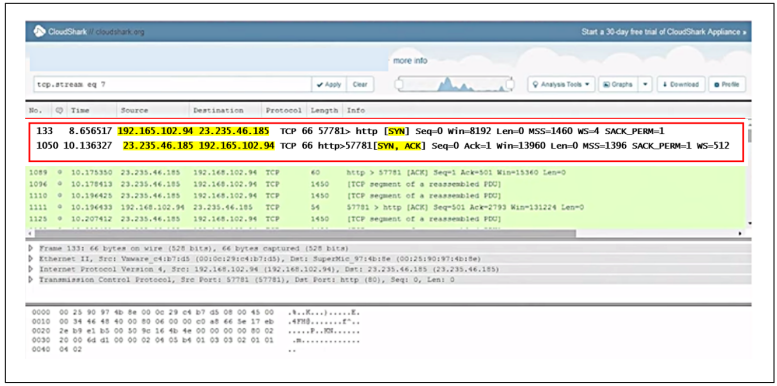


Figure 3-11. Network traffic analysis: pinpointing server latency using Cloudshark

Component-Level Analysis

Following the investigation of response anomalies, it is useful to undertake some detailed analysis of the client-side components of the target page(s); for example. each individual page in a key revenue-bearing transaction. This is sometimes termed *static* (as opposed to dynamic) analysis.

Raw data from representative individual tests (in controlled conditions) should be downloaded and examined. It can be particularly useful to compare with a similar high-performance reference site (possibly a competitor). Such analysis should include consideration of both individual components and network PCAP traces.

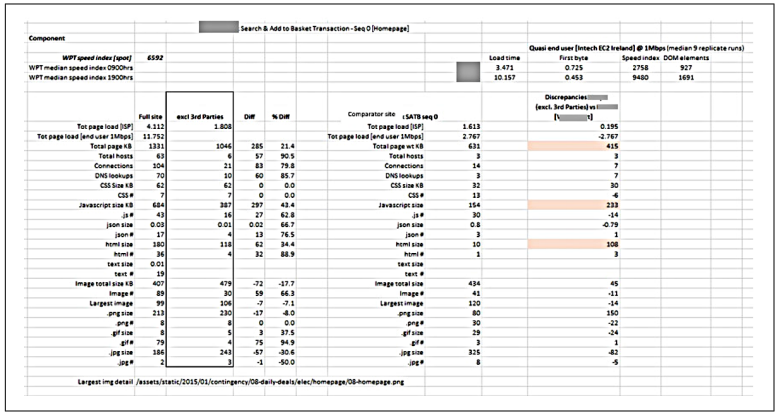


Figure 3-12. Individual component/competitor comparison; high variation highlighted

Client-side logic (JavaScript)

In-browser logic is typically introduced using JavaScript, a lightweight programming language developed for this purpose. To be most effective, screening and optimization of JavaScript components ideally should be undertaken by the developers who originated the code—or, at least, with a strong familiarity with such programming and the application itself. In some cases, such as the incorporation of third-party content into the site (e.g., the use of gambling games within betting and gaming sites), direct intervention is not possible. In any event, the following should be considered. It may be helpful to use the parameters below as a checklist when reviewing the functional capabilities of screening tools.

Absolute number of scripts and their blocking behavior (if any)
: Interacting with multiple scripts introduces a network and back end infrastructure overhead. It is often useful to seek to concatenate scripts where possible.

Download source, number of domains
: Minimizing the absolute number of domains and third-party origin—particularly if very distant—likewise reduces traffic overhead.

Code size
: Compression, minification. Network bandwidth overhead reduction.

Code coverage.
: Are the same calls being made by multiple scripts? Ideally, any query or manipulation should be undertaken once and reused.

.js total duration versus execution time
: Long-duration scripts should be examined to understand which calls consume the most time overall and why—for example, interrogation of every Domain Object Model (DOM) element on a site is very time- and resource-consuming, especially in sites with large numbers of such elements (see Figure 3-13).

CPU overhead
: This is particularly important if delivered to limited capacity mobile devices. A combination of poor characteristics such as those above can result in severe impact on users with entry-level devices. It is worth noting the perhaps obvious point that some markets, particularly those in emerging economies, have a far

higher penetration of such devices than advanced markets (see Figure 3-14).

Where practical, it is often instructive to analyze and compare equivalent pages/functionality on competitor sites, particularly those with a superior response performance.

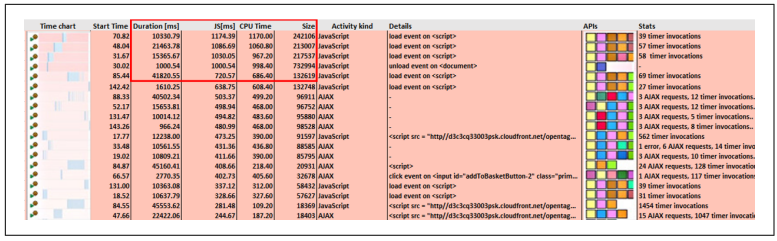


Figure 3-13. JavaScript comparison; note discrepancies between execution time/ total duration, CPU overhead, and size (dynaTrace AJAX Edition)

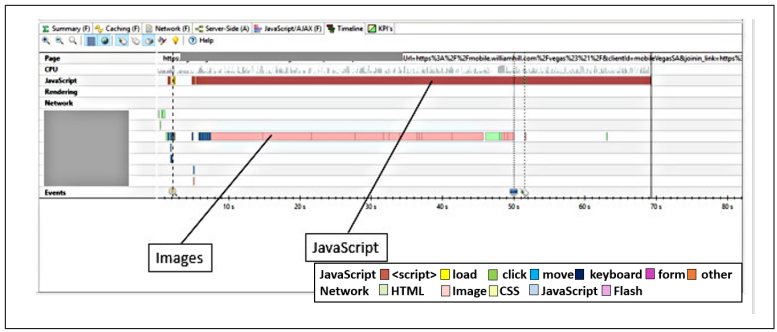


Figure 3-14. Mobile response; timeline report; note-intensive JavaScript processing (Chrome Developer Tools)

It is now useful to focus on individual "rogue" scripts. As always, drill down systematically from high level to the most granular supported by your chosen tooling (see Figure 3-15). Where is the time going?

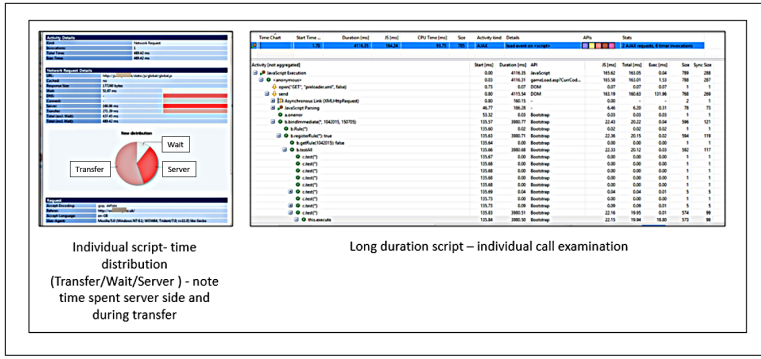


Figure 3-15. Examination of individual JavaScripts

Images/Multimedia

Images and multimedia content often represent a huge proportion of the total payload of a page. This is a very fertile area for enhancement, and also for the application of effective governance of acceptable content specification. Such governance is achieved by explicit business controls and processes (e.g., maximum size, format specifications), "flight manuals," and the like.

Within the application, it is important to avoid bad practices such as HTML-based scaling. Content format should be considered, both in terms of relevance to the image type—icon, animation, transparency, etc.—and the recipient user device. The former is covered well in the relevant chapter of Steve Souders' *Even Faster Web Sites* (O'Reilly, 2009) though unfortunately, the SmushIt tool is no longer available. Some visibility of the effect of optimal compression (at least as far as *.png* images are concerned) can be gained from using other open source tools such as pngcrush or png gauntlet.

Efficient image handling by some devices is facilitated by using an appropriate format; for example, WebP images for Android mobile devices will save some 30% of total size compared to standard JPG codecs, with quoted delivery efficiency of around 10% at equivalent image quality. Compression is important, as is removal of superfluous (nonvisual) image data—such as the date—so-called "image metadata." This is handled automatically by many image enhancement programs. As a general rule, some of the most dramatic savings are delivered by ensuring that *quality* is the minimum acceptable rather than best possible. This is a subjective decision, but well worth exploring, especially for small-format mobile devices.

Having determined and stored a set of ideal images, delivery can be managed automatically by reference to the visitor browser user string. These services are offered as options from established major content delivery vendors. A number of specialist companies also exist. The imgix site is worth exploring, if only to examine the effect on overall size of changing specific parameters within their (foc) sandbox. A Google search will reveal other (open source) tools. With regard to monitoring multimedia streams, it is worth mentioning that several major APM vendors are planning this capability as an extension to RUM functionality in their downstream roadmaps.[2]

Figure 3-16 from a Meet the Team page on a mobile site illustrates the dramatic reduction in size with little loss of clarity. Further reductions are certainly possible, but this illustrates the point.

Figure 3-16. Image optimization

Readers interested in the specifics of image optimization (which is a fairly large and detailed area) are referred to Ilya Grigorik's excellent detailed treatment of the subject.

Additional Considerations

The previous examples highlight some of the most fertile areas for potential optimization. Many others exist, such as cache/compression handling, third-party content interactions (including daisy-

2 Author, personal communication

chaining behavior by introduced affiliate tags—a potential security, as well as performance, issue).

Affiliate Content

Poor governance of "Performance by Design" practice include the presence of multiple/outdated versions of affiliate tags or jQuery libraries, and sites with excessive numbers of DOM nodes. Although a not infrequent cause of performance inefficiency, the latter is a good example of a finding that is *not* amenable to immediate change, as it requires a fundamental rebuild of the site.

The blocking behavior of stylesheets and fonts is worth considering. In the case of fonts, this is true particularly if your user base has a high proportion of Safari users due to this browser's poor response to missing content; it will wait for an indeterminate period for missing fonts rather than rapidly switching to a default.

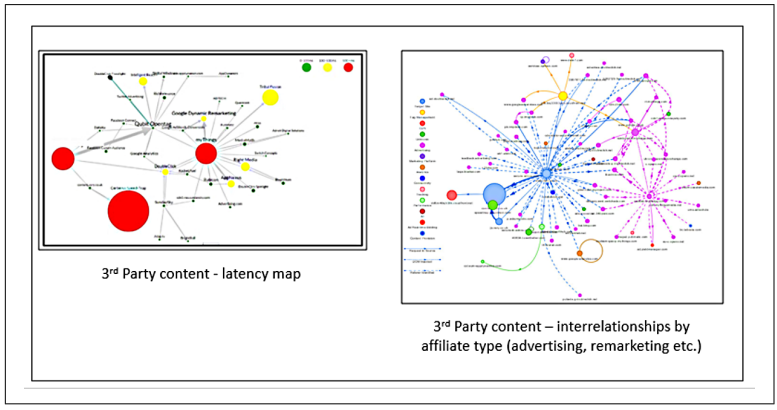


Figure 3-17. Affiliate content: two aspects of interaction (Left: Ghostery. Right: Requestmap by Simon Hearne of NCC Group)

Other third-party performance examples are shown in Figures 3-18 and 3-19.

Request details											
#	Resource	Content type	Start offset	DNS Lookup	Initial connection	SSL negotiation	Time to First Byte	Content Download	Bytes Downloaded	Error/Status Code	IP
87		text/html	8.587 s	689 ms	456 ms	-	198 ms		0.4 KB	302	64.12.106.9
85		-	8.189 s	660 ms	59 ms	-	180 ms		2.3 KB	302	193.0.160.238
88		text/html	6.025 s	611 ms	38 ms	-	45 ms		1.0 KB	302	74.125.24.154
130		text/html	11.878 s	541 ms	81 ms	959 ms	95 ms	179 ms	36.1 KB	200	-
123		text/html	10.946 s	490 ms	32 ms	97 ms	45 ms	8 ms	0.4 KB	302	-
55		image/gif	4.744 s	480 ms	31 ms	-	63 ms	-	0.4 KB	200	74.125.138.94
119		text/html	10.955 s	466 ms	31 ms	-	63 ms	3 ms	3.3 KB	200	96.7.225.98
127		text/html	11.691 s	437 ms	253 ms	-	186 ms	-	0.6 KB	200	96.7.225.98
57		-	4.776 s	420 ms	48 ms	-	69 ms	-	0.7 KB	302	159.9.37.102
70		application/javascript	6.892 s	419 ms	29 ms	-	44 ms	18 ms	12.1 KB	200	96.7.225.98
129		image/gif	11.751 s	417 ms	335 ms	-	128 ms	-	0.5 KB	200	54.230.49.25
44		image/gif	3.643 s	403 ms	329 ms	-	305 ms	-	0.6 KB	200	199.255.32.35
126		-	11.669 s	390 ms	253 ms	-	133 ms	-	0.7 KB	302	173.241.240.220
108		-	9.973 s	358 ms	267 ms	-	223 ms	-	1.8 KB	302	193.0.160.238
92		-	8.784 s	355 ms	174 ms	-	369 ms	-	0.7 KB	302	217.163.21.35
119		-	10.953 s	334 ms	32 ms	-	36 ms	-	0.7 KB	302	92.123.72.22
71		text/javascript	6.936 s	317 ms	57 ms	-	67 ms	-	1.1 KB	200	204.93.252.69
24		application/xjavascript	1.917 s	293 ms	44 ms	-	84 ms	206 ms	107.7 KB	200	54.230.30.206
94		application/xjavascript	9.213 s	293 ms	123 ms	-	55 ms	11 ms	8.1 KB	200	185.31.18.175

Figure 3-18. Third-party affiliate tag content—DNS response (WebPageTest.org)

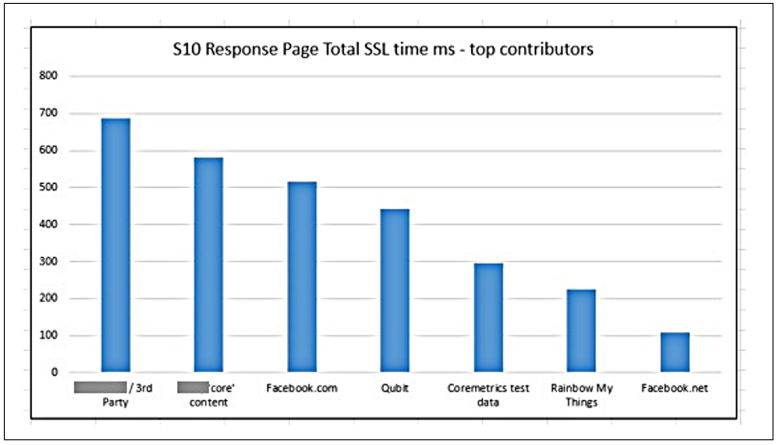


Figure 3-19. 24-hour mean SSL response by origin

An Optional Allied Activity: CDN Performance Assurance

Although not strictly within the remit of frontend optimization, Content Delivery Network (CDN) efficiency often has a major influence on performance, particularly for sites with a highly distributed global customer base and/or heavy content loads.

CDN usage is extremely pervasive, both for core content and by third parties. They can also represent a significant ongoing investment overhead. Should you wish to understand CDN performance, a useful first step is to screen your target site with a (free) tool such as CDN Planet's CDN Finder.

This will, in most cases, display a list of CDNs detected by domain. CDN technology is very powerful, but needless to say, it does not have miraculous powers. Like any other tool, it relies on correct initial configuration of the CDN itself and the cache settings of the accelerated site. Individual CDNs do vary in efficiency, particularly between global regions. For all these reasons, it is worth undertaking a CDN performance assurance test, providing that you have suitable tools at your disposal. Today, this limits you to a couple of vendors, but the situation may evolve.

The functionality required is the ability to test from a distributed network of *consumer* devices. ISP-based testing is of limited use for this purpose (for reasons that I don't have space to go into). Although in an ideal world ongoing assurance testing (linked to a Service Level Agreement) is beneficial, in practice a limited test of 24- or 48-hour duration will pick up any gross issues.

Two aspects are useful for FEO screening purposes (Figure 3-20 illustrates both):

- If it is possible to set up testing directly against the origin content (bypassing the CDN)—this will depend on how the site is coded with regard to the CDN—set up parallel tests from end-user locations in relevant geographies: one test navigating to the origin servers, and the other to the local (CDN) cache. The discrepancy between the response values obtained is effectively what you are paying the CDN for. In Figure 3-20, an average acceleration of 77% was delivered during the test period. For ongoing tests (e.g., for operations dashboards, etc.), it is easier to visualize using average line traces rather than scattergrams.
- Using reverse IP lookup, examine the origin location for the CDN content. Bear in mind that CDN delivery is dynamically optimized for performance, not origin location. However, such inspection can pull up examples of poor configuration where present. Figure 3-20 is normal, although the range of origin locations (delivering content to an exclusively UK user base) is interesting.

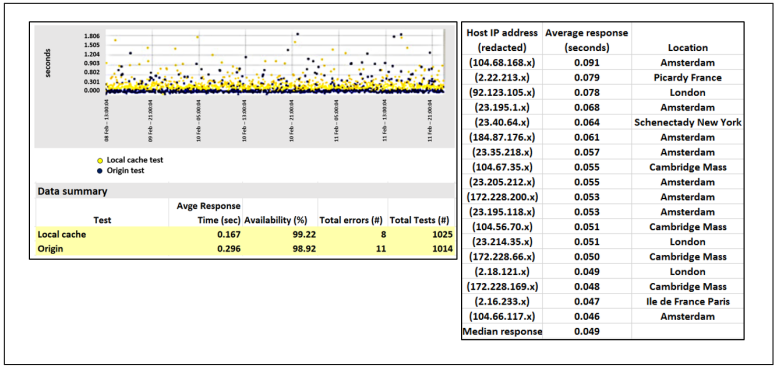


Figure 3-20. Aspects of CDN performance (Credit: Gomez)

Backend interactions

Although outside the scope of this book, it may be that performance "gating" factors are found associated with interactions with the core delivery infrastructure, like database lookups or other web services calls, for example. The more holistic aspects of Application Performance Management (APM) tooling can be of considerable value here in tying together backend issues with outturn performance.

Emerging Developments

This section provides input as to monitoring approaches and modifications suggested by various cases, driven by advances in technology increasingly adopted within target applications.

The following are briefly addressed in this section:

- Single Page Applications
- Applications incorporating server push and/or service worker
- HTTP/2
- Microservice-based applications
- Bots

Single Page Applications

So-called Single Page Applications (SPA) are becoming increasingly common, either as a complete application (as the name suggests) or as an element of a larger "compound" application. SPAs are charac-

terized by their use of client-side JavaScript frameworks (such as Angular or React). They permit dynamic extension of HTML and leverage the computing power of modern user devices.

The issue that SPAs present from a monitoring perspective is that they minimize the network interactions between user device and origin infrastructure. The historic page-based download paradigm (and dependency) doesn't apply. This presents a problem for traditional synthetic monitoring, given that they are based on capturing and analyzing just that over-the-wire interaction.

User:site interactions (termed *soft navigations*) and data delivery are independent of the standard W3C navigation API flags (DOM Ready, onload, etc). Many interactions occur entirely within the client device.

Although some nuances can exist depending upon the detailed design of applications—unless a user interaction (e.g., button click) is reliably associated with a network request—the primary (but important) value of synthetic monitoring in this use case becomes the monitoring of *availability.* This key metric is unavailable to passive (site visitor) based tools, for obvious reasons.

Any interactions that are directly linked to a network call can (in most synthetic monitoring scripting utilities) be specifically "bracketed" and their response patterns examined. Otherwise, monitoring best practice requires the use of RUM instrumentation.

Unfortunately, not all RUM tools are created equal, so if it is likely that you will be squaring up to SPAs, it will be important to check (and validate) that your RUM tool (as an extension of your APM tooling or otherwise) offers the granularity of recording required. If not, an alternative (assuming the APM vendor cannot provide realistic comfort regarding their roadmap) may be to integrate a standalone RUM that can offer such functionality. At least one current product (at the time of writing) has been specifically modified to meet the SPA use case. However, this is an evolving situation of direct business relevance. Others will undoubtedly follow.

HTTP/2-Based Applications

The evolutionary specification of HTTP/2, a formalization of Google SPDY, has been available for some time. Adoption is now

reported to be rapid, and this rate is expected to further increase with progressive server adoption.

HTTP/2 provides a number of transport efficiencies relative to HTTP/1.x. These include request multiplexing (i.e., effective handling of multiple requests over the same connection), compression of head components, and other design interventions to avoid multiple retransmission of head metadata.

These changes deliver considerable advantages, certainly theoretically, in sites with large numbers of element requests and those involving delivery to users in high-latency conditions, although the jury is still out to some extent. They also make it necessary to be aware of changes in interventions formerly regarded as best practice for optimized performance.

Domain sharding, which was formerly adopted to increase the effective number of parallel connections, becomes an antipattern. Domain sharding involves the risk of request failure and consequent retransmission, particularly in conditions of limited connectivity (mobile delivery in rural locations and countries with poor internet infrastructure). It impacts the inherent HTTP/2 efficiencies of header compression, transmission optimization, and resource prioritization possible over connection to a single domain. It does not present monitoring or analysis challenges per se, and can form part of optimization recommendations.

Content concatenation, the most prominent usage of which is in image spriting but which may also be applied to other content, aims to reduce the number of roundtrip requests. This has, however, the disadvantage of forcing refresh if any part of the grouped content changes. Revised best practice, driven by the transmission efficiencies inherent in HTTP/2, directs reduced individual object payloads and essentially a more granular management of content at individual element level. This, for example, supports more appropriate cache setting with regard to the specifics of particular objects.

Inlining, the incorporation of content (e.g., JavaScript) within highly prioritized download components (e.g., HTML), was formerly adopted to accelerate delivery of required content while minimizing the requirement for round-trip journeys and delays due to differential content-type download priorities by HTML. It had the disadvantage of preventing individual caching of the inlined content. The recommended best practice replaces inlining with server-push-

based delivery, thus supporting both progressive content delivery and more granular cache management.

It should be noted that, apart from increased adoption of server-push interactions (see "Persistent Browser Interactions" on page 62), these changes involve modification of FEO interpretation and recommendation, rather than impacting monitoring practice.

Persistent Browser Interactions

This section draws together a number of developments that, if unmanaged, will introduce serious distortion into monitoring outputs using "traditional" tools. The Beacon API, a useful approach to gaining understanding in these situations, is highlighted.

Persistent duration *server:client* interactions are a core facet of modern applications. In certain cases, this is driven by the nature of the application itself (e.g., delivery of live-update betting odds). Other drivers are the leverage of HTTP/2 efficiencies (see "HTTP/2-Based Applications" on page 60) and the development of network-independent mobile WebApps (otherwise known as Progressive Web Applications [PWA]).

PWAs effectively coexist with native mobile applications. They incorporate local device caching, and store and forward capabilities that enable usage in unstable or off-network conditions. PWAs utilize service workers. These replace the limitations of former AppCache-based approaches. They are event-driven, and facilitate great flexibility in content delivery. Service-worker capability offers many attractive advantages in the creation of more business-centric mobile-device-based interactions.

The challenge to historic monitoring practice is that long duration connections (of whatever type) distort the recorded page-load end point in traditional synthetic monitoring tools. This must be identified and corrected for; otherwise, incorrect performance inferences may be drawn, particularly in terms of recorded response variation.

Fortunately, identification of server push or other long-duration interactions is usually obvious from inspection of standard waterfall charts. Correcting for it in an elegant manner is more difficult. If we ignore the (semi-proprietary) validation approaches incorporated within the scripting tools of certain synthetic monitoring providers —as they are not widely adopted—arguably the best approach to

synthetic testing is simply to identify and then filter out the long-duration calls. Although somewhat of a blunt instrument, it does get around the problem.

A more elegant approach, based on RUM analysis, emerges with the availability of the new sendBeacon API, the syntax of which is:

```
Navigator.sendBeacon(url,data);
```

Use of this call enables granular instrumentation of application code to specifically record response to events. It should be noted that this is newly released (at the time of writing), so it is likely that reliable cross-browser support is unlikely to be complete. However, I understand that the leading-edge performance team at *Financial Times* in London reports effective use of this API in production conditions.[3]

```
1  window.addEventListener('unload', logData, false);
2
3  function logData() {
4      navigator.sendBeacon("/log", analyticsData);
5  }
```

Figure 3-21. Example instrumentation code using the sendBeacon API

Internet of Things

Sensor-based applications, collectively known as the *Internet of Things (IoT)*, have been slowly evolving since Coca-Cola introduced the self-reordering dispensing machine more than 40 years ago. It is now in danger of becoming one of the most hyped areas of new technology. Certainly, actual companies are now trading (in the UK, Hive and Nest to name but two). Regardless of whether the app is controlling your heating thermostats, reordering the contents of your fridge, or (in the future) ordering up a driverless car for your commute to work, it is important to be able to understand and validate performance in objective terms. Figure 3-22 shows an example of a mobile device-based controller and associated API.

Unfortunately, IoT networks are fairly closed systems. As such, they are not accessible to scheduled synthetic external testing. However,

3 P. Hamann, personal communication

at the end of the day, it's only device monitoring, so a strong crossover with traditional end-user monitoring exists. Two approaches are possible:

- Instrument the mobile application used to control the system
- If available, monitor via the API or, if available (typically in APM tooling), a WebServices-based gateway to the application in question

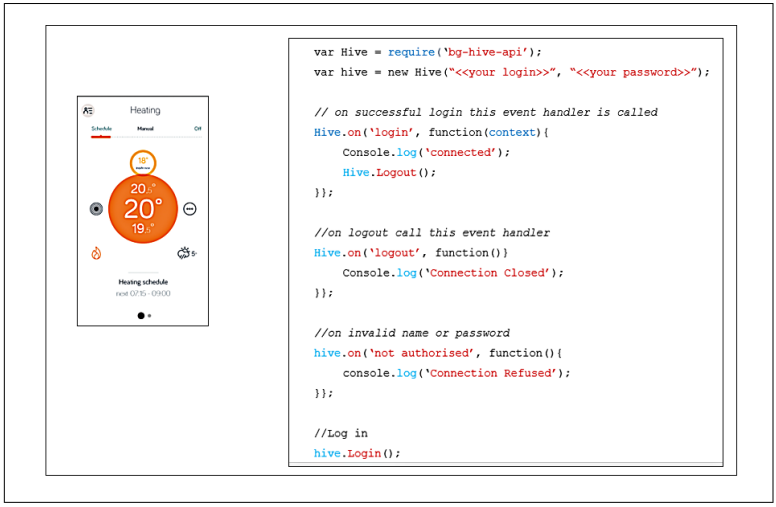


Figure 3-22. IoT application control panel and (third-party) API code (HIVE Home)

Microservice-Based Applications

The primary development and/or extension of applications based on microservices—discrete functionality containerized elements—is becoming very popular. Arguably, this is being driven by the popularity and availability of open source platforms, particularly Docker, though alternatives exist.

The pros and cons of microservices adoption are outside my experience and the scope of this material. Suffice it to say that despite the ownership advantages of highly granular functional elements from an agile development perspective, microservices-based applications deliver an additional layer of integration and management complexity from an operations perspective. Performance understanding should be considered from both a backend and external approach.

From the point of view of the containers themselves, the major APM vendors are increasingly including specific support for these technologies. Currently, given the market dynamics, specific support starts with Docker, although other platforms will be explicitly supported moving forward. The extent of visibility offered by the various APM tools does vary, although it is likely that your choice will be made by other considerations (and therefore you will "get what you get" with respect to container performance visibility). Figure 3-23 shows an APM based example of visibility of microservices components (in this example, using Docker).

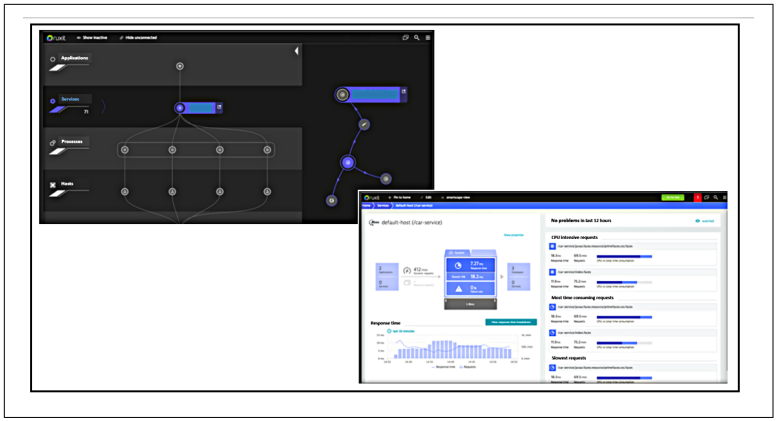


Figure 3-23. Microservices container monitoring (Example: Ruxit)

In terms of external monitoring practice, the core change is not the high-level approach/tooling mix, but rather the importance of ensuring that poor performance of core services and/or module interactions are targeted, such that interventions can be made rapidly. This is particularly apposite given that the nature of testing and preproduction environments is such that it is likely that issues will arise that only emerge post release-to-production when the application encounters real-world load and interaction complexity conditions.

The take-home message should therefore be to monitor with underlying services in mind. This implies a sub-page monitoring approach. Greater granularity of monitoring can be achieved, by, for example, (with synthetic tooling) scripting transactions to bracket key in-page interactions (reported as step timings), and (with RUM) using additional timing markers/beacons to achieve the same effect. Issues not specifically detected by these techniques should reveal

themselves by changes to traffic flows and user behavior. These are best detected by cultivating an approach to web analytics reports that is both absolute and intuitive.

Bots

Although not strictly associated with FEO, a few words on bots are relevant to the consideration of third-party-related performance constraints. *Bots* (or *web robots*) are automated site interactions. Although the majority (ranging from search engines to synthetic testing and price aggregation) are not malicious in intent, they represent a huge proportion of total site traffic—over two-thirds for typical retail sites, for example.

Figure 3-24 below is a particularly clear illustration, derived from IP traffic analysis on a UK retail site, of the predominance of bot traffic (bottom histogram) versus customer traffic (center histogram).

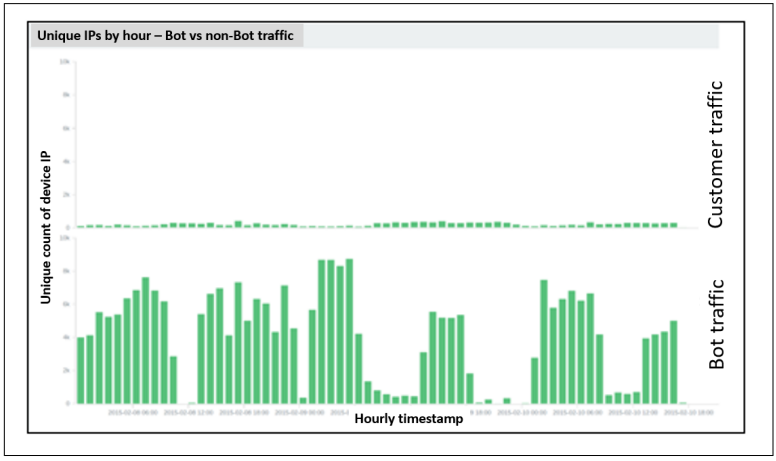


Figure 3-24. Global car rental site; UK traffic by unique IP per hour; total versus customer traffic

This level of bot traffic represents a significant economic cost, both in maintaining otherwise unnecessary infrastructure and in reducing the effective capacity overhead of the site (and therefore its ability to benefit from peaks in real-customer traffic). These benefits can be extremely significant. One of the retail clients of Intechnica—the performance consultancy where I worked for several years—was able to reduce its IBM license requirement for WebSphere Com-

merce Suite from five to three cores. This resulted in a significant saving in annual license costs.

Unfortunately, bot effects are not simply confined to generating excess traffic. So-called "bad" bots have a range of negative effects, from inadvertent inefficiencies due to poorly written code, to spam, malicious hacks, and high-volume Denial of Service (DDoS) attacks. According to the Anti-Phishing Working Group (report, Q1-Q3 2015), over one-third of all computers worldwide are infected with malware.

Various approaches to mitigating bot-traffic effects are possible. These include:

- IP blocking
- CAPTCHA (largely regarded as compromised)
- Multi-parameter traffic *fingerprinting*
- Bot *honeytraps*

From the point of view of performance practice and FEO, bots are an indirect consideration but one that should be considered when making recommendations regarding overall performance enhancement. Seek to quantify the extent of the problem and identify potential interventions. These are likely to depend upon the economics of the threat and existing relationships. They can range from specialist target solutions, security extensions to firewalls, value-added options from CDN or other performance vendors, to focused integrated traffic-management solutions.

Securing Gains with Ongoing Monitoring and KPI Definition

Ongoing monitoring and key performance indicators (KPIs) are important facets of a complete end-to-end approach to quality assurance in this area. Although outside the scope of this book, interested readers are referred to my extended treatment in *The Art of Application Performance Testing* by I. Molyneaux (O'Reilly); Appendix B.

CHAPTER 4

Building a Performance Culture in the Organization

"Ad augusta per angusta" (to high places by narrow roads)
—Victor Hugo

This final chapter considers aspects of performance from a business management perspective. Although you may provide oversight, ultimately the details of monitoring and analysis will be performed by analysts and other IT professionals of one sort or another. Two areas remain where line-of-business managers can have a beneficial influence. One is tactical; the other strategic.

Tactically, if preliminary monitoring versus your key competitors indicates a significant performance deficit, or if you are in the happy position of being able to monetize performance (see Chapter 1) and such monetization indicates a major impact on digital revenue, then you may wish to invoke external assistance in terms of third-party performance-enhancement solutions. The strategic area is that of evolution of a *performance culture* within your organization. This is easier said than done, as any cultural change is a major undertaking. However, the real-world benefits for organizations adopting such a path are large. I will address this longer-term approach later in the chapter.

NOTE

Significant UK examples of an explicit performance orientation in both digital and broader cultural areas include:

- The Guardian and Financial Times in the media sector
- Laterooms in the travel sector
- Next in the retail sector

For the FT, reference "Performance management: five priorities for designing a system that aligns with IT article". Each of these companies have experienced broad benefits,[1] some intangible, although The Guardian is currently website of the year, which is proudly displayed on their masthead, shown in Figure 4-1.

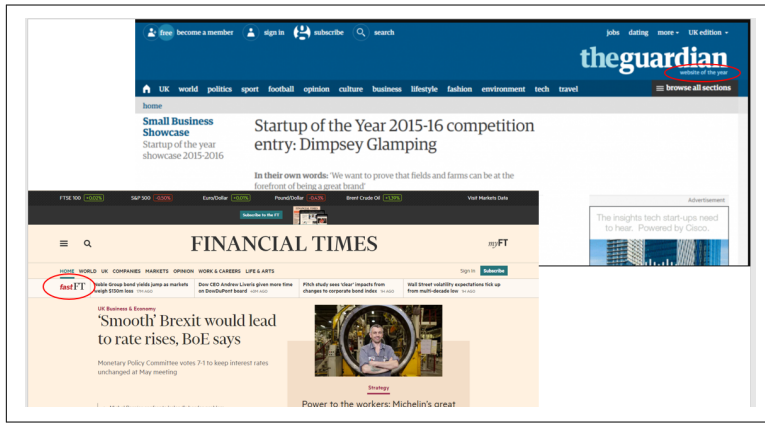


Figure 4-1. The fruits of performance orientation; two media-sector examples

Reverting briefly to tactical approaches: depending on circumstances (e.g., global distribution of customers), a number of performance interventions can be "bolted on" to an existing application infrastructure to enhance performance. They do work, although some are somewhat based on the principle that "in a hurricane, even a turkey can fly." If the fundamentals of your application are poor, then such interventions (CDNs, etc.) can become very expensive.

1 Author, personal communication

However, it is to be hoped that applying some of the principles outlined earlier will enable some optimization of the base situation. Differences obviously exist in functionality (for example, CDNs vary widely in certain characteristics such as regions served, points of presence per region, etc.) as well as price. That being the case, it is important to undertake a structured proof-of-concept trial prior to adoption.

I will now go on to consider building a performance culture in more detail.

Building a Performance Culture

End user monitoring and analysis are important and should pay dividends, but they are not ends in themselves. They must compete for resources with many contending priorities—OPEX control (e.g., headcount freezes), exotic marketing campaigns, and many others. All too often, they can lose out, despite the associated business risk from neglecting this area.

The most effective businesses manage to make digital performance part of their culture. I conclude with a few thoughts on this.

The Bricks-and-Mortar of Performance Culture: Some Practical Ideas

In the immortal words of Michael Brunton-Spall when opening the 2015 Scale Summit event in London: "Technology is easy; people are hard."

Many people have observed three elements in achieving successful delivery of IT performance excellence: people, process, and tooling. The following are some thoughts on the people part of the triad that may assist others treading this path.

The Nature of the Problem

Every group has its culture, but in large organizations, culture often serves to counteract the requirements of a performance focus. Particular points that might resonate with readers are internal politics, siloed operations, and in particular, a conservative mindset whereby the business is resistant to any change in IT systems due to a perception that change equals risk. These features of most large organiza-

tions by virtue of their size are exacerbated in conditions of high growth and/or acquisition, with the imperative to subsume all outliers (acquired companies, small teams) into a common mothership mentality. This is often underpinned by a desire to impose the parent's brand values, and to promote and impose their management across the empire. In this environment, small IT teams can feel trapped within a "don't touch anything" mentality, with any change moderated through endless committees, and any concept of enterprise or initiative stifled.

Sound familiar? So what strategies are available for seeking change from within, rather than joining the mass exodus towards superficially greener pastures?

Requirements for Change

From the point of view of the IT teams responsible, to enable effective engagement in the delivery of application performance, the company culture has to engender four key factors, assuming that core skills around monitoring, unit, and performance load testing are available.

Assuming that it is supported by the wider culture, the bases of success are (what I term) the *CEFO elements*:

- Cooperation
- Empowerment
- Freedom of action (within agreed boundaries)
- Objectivity

These combine to provide the final component: visibility. Visibility is the tangible expression of performance-centricity throughout the organization. Such expression will typically have many forms but should be apparent to those outside the business (e.g., visitors or customers) through, for example, logo straplines or heads-up dashboards, as well as to employees and managers via balanced scorecards or incentive schemes.

Given that CEFO conditions will not magically arise, it is necessary for performance to become a central tenet of the overall company culture. The difficulty, supported by a mass of literature on the subject, is that changing a culture, particularly in large organizations, is extremely difficult. However, it is not impossible given the right

conditions. The following are ideas gleaned from successful attempts.

Building Blocks of a Performant Culture

Interaction with companies (such as those referenced above, and others ploughing the same furrow) reveal several core building blocks for a successful cultural change outcome. They are listed here, though others may apply to your circumstances, such as managing disparate international cultures and/or business unit autonomy.

Culture is for life—not just for Christmas.
: Less flippantly, cultural transformation is a serious, core, long-term undertaking. It is not, and can never be, a tactical exercise.

Get senior buy-in.
: It is probably no exaggeration to state that cultural change can never be effected from the ground up. Senior buy-in (preferably ownership) is essential. To achieve this, consider:

  - Presenting the problem (for example, unperformant applications) in terms relevant to the audience. Senior management are not interested in server throughput, but they are likely to respond to loss of market share or revenue, and concern about share value.
  - Visualize the issue. Tools such as WebPageTest's side-by-side videos are great at driving home a competitive deficit.
  - Support with objective data, especially from end-user testing (international markets, native mobile applications, etc.). Trending is particularly effective when data is available, and helps make the point that this problem is not going away.

Build awareness across the company.
: Performance projects with five-letter acronyms are probably too simplistic, as are anodyne "ten values" statements. However, if the company goals can be succinctly stated and promoted by Level 1 executives, then half the battle is won. Within these, make statements referenceable rather than "motherhood and apple pie."

Make performance an explicit, visible goal.

IT can crystallize these as specific KPIs, which are particularly effective if delivered across heads-up dashboards in key areas. Within dashboards, RAG (red-amber-green) outputs or speedometer dials are likely to have the most impact, especially if associated with revenue (e.g., orders per hour versus site response).

Take a structured, process-based approach to adoption.

A useful idea suggested by one of the companies alluded to in recent sections (borrowed from the late, often-lamented Steven Covey) was to require each subordinate department/level within the organization to state how they would achieve the published Level 1 goals. These statements should incorporate SMART (specific, measurable, actionable, realistic, timely) objectives. Within IT, product owners can be key to the solution by ensuring that nonfunctional requirements are core to the specification, tested (and budgeted for) as part of an integrated continuous-integration DevOps process.

Reach out to the whole company.

As an example, suggestion-box applications have a long history, stretching back at least 25 years to Archie Norman's "Tell Archie"-led transformation of ASDA, although there are many others. Some form of incentive for the submission of particularly useful ideas is beneficial, although prizes do not necessarily have to be tangible. Advocacy can be a reward in itself by associating an individual or team with business success.

Take care when crossing the boundaries of the organization.

It is important to avoid partnership-based clashes (in the jargon, *acculturation*). If possible (and it is always possible at some level), it is important to undertake some form of cultural screening of partners. This is a field where (at last) there are a few companies who will add value. However, seek to require objective (as opposed to narrative-based) comparison. The latter is more relevant to the descriptions of far-flung indigenous populations in a different era. Even without such assistance, the performance-centric company rapidly becomes attuned to disparities in other organizations. Seek to undertake informal due diligence specifically around working culture. Signs and artifacts (even matters as superficially trivial as tea-room culture) within potential partner organizations can reveal a lot.

Recruit and structure with intention.

One of the features of company cultures (be they beneficial or toxic in the eye of the beholder) is that they tend to be self-perpetuating, in that those who like/can cope with (or derive some personal advantage from) them stay, and others leave. This is one of the reasons why change in this area is so difficult. However, any organization has a degree of staff churn—retirements, changes in circumstances, etc. This can and should be harnessed as a change agent. Thus, in addition to the cultural memes adopted (and referenced elsewhere in this chapter), it is beneficial to:

- Recruit core people willing to explicitly embrace performance, and who can reference such an orientation at interview.
- Consider an entrepreneurial digital-development role (separate from core IT). Such a role will need to be publicly supported by Tier 1 management if it is not to be suppressed by "the blob"—the forces of the *ancien regime,* particularly those of departments such as IT, which tend to be resistant to such change.
- Consider change agents such as ITIL, which can have positive benefits such as changing "operations" to "service," but beware the elephant trap of bureaucracy. Who will have ownership of such introductions?
- *Every* level in the organization should have ownership. This enables organic development, rather than a perception of imposition from above.

Above all, note that the real values of an organization are expressed through who gets promoted and explicitly rewarded—not on what lip service is paid.

Further Reading

For anyone interested in reading further about the subject, I recommend the following resources (selected from the vast academic literature):

Cultures Consequences, 2nd ed, by Geert Hofstede (Sage, 2003)
: An academic study of some of the dimensions of corporate culture, derived from Hofstede's several decades of work at IBM. Old, and based on national cultural dimensions, but classic.

In Great Company, 2nd edition, by Q. Jones et al. (Human Synergistics, 2011)
: A practical guide, based around real-world case studies.

"The big debate: performance management", published in The Guardian
: Discusses performance management more broadly within the UK public sector. Although not concerned with digital performance, it contains references to some useful sources of information for those wishing to adopt a more holistic approach to performance throughout an organization.

Conclusion: Everything Changes, Everything Stays the Same

In conclusion, the rate of change in the variety of end-user applications continues to increase. Such change is inevitably associated with performance challenges for two reasons. One is that applications developed for a particular usage and technology environment (e.g., PC browser/standard HTML) continue to be used while behaviors, technologies, and devices change (e.g., progressive applications delivered to mobile devices). The other is that developments in applications themselves render tools and techniques of understanding redundant (consider Single Page Applications and synthetic monitoring, discussed in Chapter 2, as examples). Some aspects of new technologies may be black-boxed as far as monitoring and analysis is concerned—at least, if without access to core systems and code (e.g., Internet of Things).

However, in the clear majority of cases, every meaningful aspect of an end-user transaction will continue to involve visible changes to the GUI (Graphical User Interface; in simple parlance, the user screen). As such, end-user performance can continue to be monitored. Network interaction will still be required, even if it's on a store-and-forward rather than real-time basis. Together, these provide the basis of frontend analysis and optimization.

Provided that there is access to developers and source code, outputting timing metrics via the range of APIs becoming available provides a robust, production-ready richness to the analysis of visitor performance experience. These can prompt intervention and provide a deep understanding of production performance. The caveats already expressed—monitoring availability, competitors, and performance in low-traffic situations (including pre-production)—still apply.

When presented with new challenges to effective practice, the cascade shown in Figure 4-2 should prove its worth.

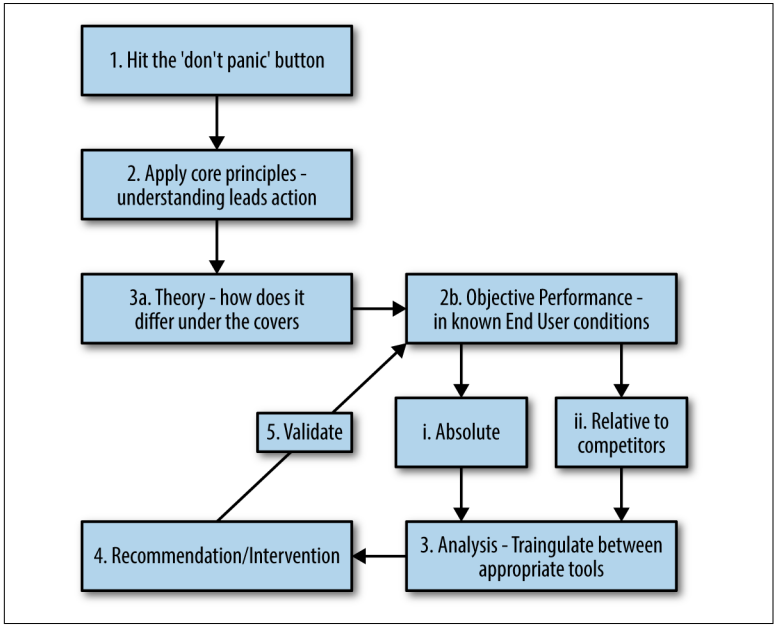


Figure 4-2. Dealing with new developments: a cascade flowmap

Final Thoughts

I will leave you with a few final thoughts about FEO recommendations. My primary objective in writing this book was to link two superficially disparate worlds—business (revenues, stock price, market share) and IT (bits and bytes, development code). In fact, as I hope I have conveyed, digital channels to market is an area where these two orbits intersect. This provides a great opportunity to share understanding and harness the resources of the business to a common goal. In seeking to achieve these benefits, consider the following imperatives:

- Link *technical* priorities to *business* drivers such as competitive revenue exposure, etc. Seek objective understanding of deficits and advantages in terms of performance (e.g., transaction speed) and customer behavior (e.g., basket size, page bounce rate, time-on-site).
- Live in the real world. What can be changed at economic cost, in realistic timescales? Turbocharging an existing application to be the fastest in the market to all users all the time is unlikely to be fiscally prudent, even if it is achievable. All the engagements in which I have been involved where this was an initial stated aim have settled for a process of progressive iterative improvement and have reaped the rewards of so doing.
- Beware major effort for marginal improvement. It is always worth standing back from a given intervention and considering the big picture—in particular, what is the projected lifespan of the application or element in question? Can this change be more readily baked in to the next-generation solution?
- Seek to deliver a combination of immediate prioritized interventions and ongoing governance/management of objectives, and set iterative goals for improvement (unless in crisis mode). These can be supported by a combination of process change and documentation. For example, what are the existing controls on adding third-party affiliate tags or huge animated GIF images (Father Christmas is a common offender here!)?
- Suggest triggers for ongoing intervention based on a combination of direct (synthetic monitoring) and indirect (web analytics, RUM) alert flags. In short, many technologies are available, and all will give you some kind of answer. Try to use the infor-

mation in this book and elsewhere as a guide to determine elegant monitoring techniques that provide maximal useable data for minimal overhead.

APPENDIX A
Tooling Types

The following reference table summarizes some core terminology around end-user monitoring.

Term	Description
External monitoring	Strictly, any regular testing (monitoring) of application response from outside the edge servers of the delivery infrastructure. This term includes both active (synthetic) and passive (actual site visitor) approaches. Synthetic/active: • Backbone (primary ISP-based testing), either from individual T1 data centers or LINX locations. • Cloud-based (e.g., for comparison of relative CDN performance). • Private-peer locations: any specific location where a vendor test agent has been installed. Typically, these are inside-the-firewall (i.e., intranet locations such as customer-service centers or branch offices), although they could include partner company sites or (in theory) IoT devices or customer test panels (e.g., VIP users of betting and gaming sites). • Synthetic end-user testing from test agents deployed to consumer grade devices. Depending upon the technology used, these can vary between "true" end users—the dynaTrace Synthetic Last Mile testing is one highly distributed example, private-peer testing, or quasi-end-user testing from consumer grade devices with artificially modelled connection speeds (WebPageTest provides a good open source example of this). Passive: • Real User Monitoring (RUM) and other synonyms; see next row.

Term	Description
Real User Monitoring (RUM]; aka passive monitoring	• The performance analysis of incoming (site visitor) traffic by detection of a variety of navigation and other response related APIs. • Performance metrics associated with other user device related information (e.g., operating system, screen resolution, mobile device).
End User Monitoring/End User Experience monitoring	As RUM (above), but note the distinction between "experience" in this sense (i.e., speed of response) and behavioral-based end-user experience tools and techniques. The latter are more associated with design-led behavior, and would include vendors such as Bunnyfoot.
Native mobile application monitoring	Monitor network interaction (e.g., ATT ARO).

APPENDIX B
Suggested Reading List

A lot of material exists. The blogosphere, vendor (and others) ebooks, and web performance meetup groups (such as the excellent London Web Perf MeetUp) are all good sources for keeping abreast of recent developments. For core reading, the following are a good start:

Title	Author	Pub	Year	Technical [1L:5H]	Notes
High Performance Websites	S. Souders	O'Reilly	2008	4	The classic. Good for core principles, but some of the detail now superseded.
Even Faster Websites	S. Souders et al.	O'Reilly	2009	4	An expanded version of the initial treatment.
High Performance Browser Networking	I. Grigorik	O'Reilly	2013	5	A comprehensive guide to this important area.
Using WebPagetest	R. Viscomi et al.	O'Reilly	2016	3	A how-to guide to one of the most popular free test tools.

Title	Author	Pub	Year	Technical [1L:5H]	Notes
The Art of Application Performance Testing 2nd Edition 2015	I. Molyneaux	O'Reilly	2015/2017	3	Currently being revised and updated prior to release of an ebook edition. A useful guide to the practice of performance load testing. Includes material relevant to end user monitoring practice (statistical analysis, key performance indicators, etc.
Effective Performance Engineering	T. DeCapua & S. Evans	O'Reilly	2016	2	A useful high-level treatment.

About the Author

Larry P. Haig has over 25 years of IT industry experience with vendors large and small, and independent consulting companies.

Larry has an MBA from Kingston University Business School in the UK, and has undertaken post-MBA courses at Warwick Business School and Harvard Business School. His original business research interest was in corporate culture and change management, which he continually tries to bring to bear on the practice of external monitoring and frontend optimization—with variable success!

Involved in corporate web deployment since the early days of the web (1995), Larry has specialized in web-based digital performance for over a decade.

In 2012, Larry was an external monitoring SME and a senior consultant at Intechnica, a performance consultancy in Manchester, UK. He took early retirement in 2016 to focus on a portfolio of IT-related business goals, although he maintains his interest in the development of performance optimization techniques. He has been a speaker at Velocity Amsterdam.

Outside work, Larry enjoys village life, long distance walking, and a variety of quixotic projects that keep him off the streets.