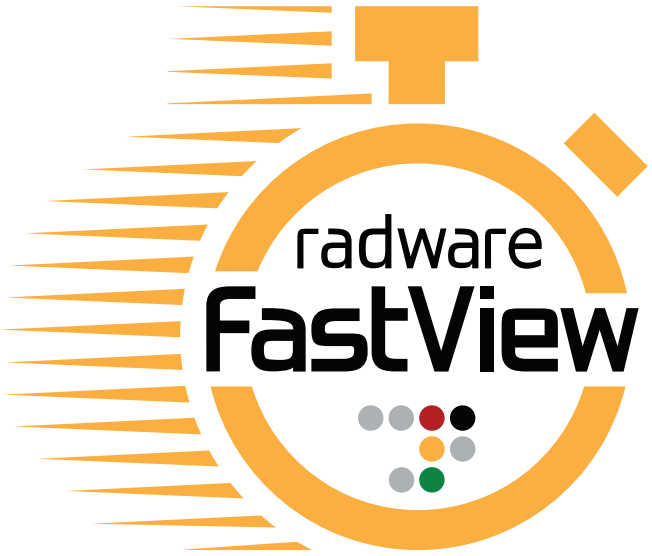
radware

Improve Online Business Performance with FastView Web Acceleration

Whitepaper



SHARE THIS WHITEPAPER

Table of Contents

Executive Summary

This whitepaper includes a brief overview of the performance challenges of modern web applications and the common and current approaches used to address them. The paper also describes Radware's FastView proprietary web performance optimization technology drilling down into its architecture and methodology.

As organizations increase dependence on web applications for both internal productivity and external communication with customers and partners, performance optimization emerges as an essential business driver. Studies have demonstrated that website performance has a direct correlation with revenue in both ecommerce and advertiser-supported applications. Users expect rich web experiences, but they easily become impatient if pages render too slowly.

Radware has developed innovative and cost-effective technology to address web performance challenges. The technology is encapsulated in an appliance that can be deployed physically as a piece of network hardware, or virtually as software that runs either onsite or as a cloud service.

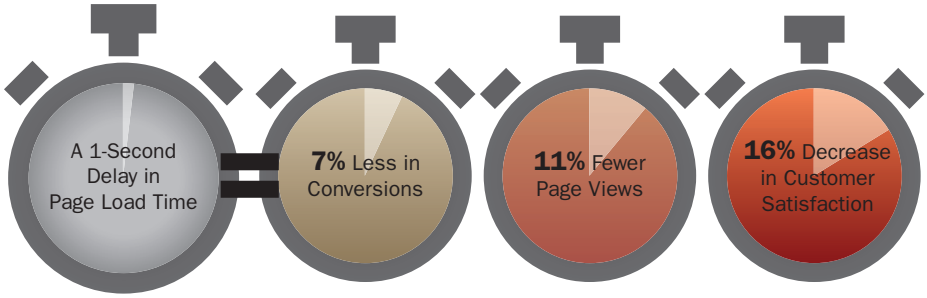


Figure 1: In dollar terms, this means that if your site typically earns $100,000 a day, this year you could lose $2.5 million in sales. Source: Aberdeen Group.

Radware's FastView is an expert system that learns the resource usage patterns of a site and dynamically applies best practice coding techniques by rewriting pages without requiring any source code modifications.

FastView analyzes usage patterns and page content, and develops a dynamic repository of rules and cached resources. By strategically modifying web pages on the fly as they are sent to users, FastView is able to improve performance significantly. The modified pages:

- reduce the number of roundtrips required to render contents
- execute client-side code in the most efficient order
- automatically preload resources that are likely to be needed for future requests
- tailor behavior to exploit the capabilities of the user's browser.

As the application code or contents change, and usage patterns shift, the rules used to modify pages automatically adjust. In addition, FastView is based on an extensible modular framework, supporting the addition of new or improved page treatments as they are developed.

Web Application Performance Benefits

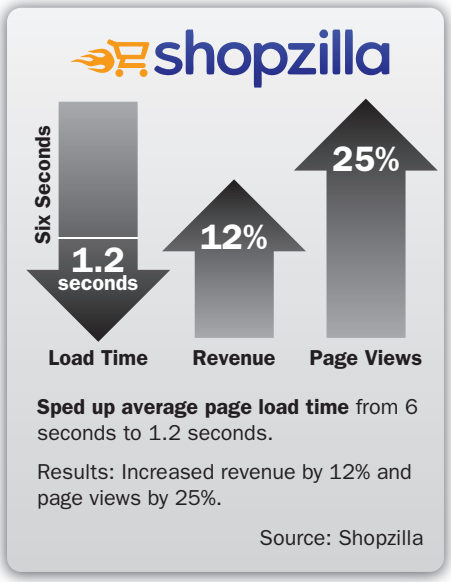
In reference to revenue generating web applications, the old maxim "time equals money" takes on new meaning. Reducing the time required to render web pages, results in more online revenue. Radware has identified four ways in which site performance directly affects business metrics:

SEO ranking

Google and other search engines allocate either a set period of time or a set quantity of data for crawling each site. Radware has performed a customer study confirming that after applying its optimizations to a site, the Googlebot web crawler was able to cover approximately twice as many pages as the un-accelerated site. Increasing the number of crawled pages directly affects rankings and ultimately traffic.

Page views

AOL conducted studies in which they measured site performance and page views. They found that visitors to the top 10% best performing sites viewed 50% more pages than visitors to sites in the bottom 10%. On average, visitors to the faster half of the sites viewed 9% more pages than visitors to the slower sites.



Sped up average page load time from 6 seconds to 1.2 seconds.

Results: Increased revenue by 12% and page views by 25%.

Source: Shopzilla

Increased conversions

Shopzilla decreased its average page load time from 6 seconds to 1.2 seconds and experienced a 12% increase in revenue and a 25% increase in page views. The faster site also doubled the number of sessions from search engine marketing and cut the number of required servers in half.

Increased revenue

Microsoft's Bing site conducted a study wherein they slowed page load times by 2 seconds. As a result of the slowdown, users ran approximately 2% fewer queries, clicked 3.75% less often, and reported a significant level of satisfaction with their overall experience. Conversely, speeding up the site by 2 seconds resulted in a 5% revenue increase.

Optimization Challenges

Modern Web Pages Place High Demands on Performance

The HTTP and HTML protocols that underlie web applications were originally developed to support transmission and rendering of very simple pages by today's standards. New technologies, such as JavaScript and cascading style sheets (CSS), as well as new media types, such as Flash videos, and more graphics-intensive pages have all added richness to the experience. However, the underlying architecture of page delivery has not evolved quickly enough to keep pace.

Although increased bandwidth has largely mitigated the impact of larger payloads, it hasn't compensated for the increased number of roundtrips required to render pages.

Many of the techniques implemented by Radware's FastView are directed at reducing the number of roundtrips required and re-ordering them to minimize page load times.

In 1995 the average web page contained just 2.3 objects, requiring only 2.3 calls to whatever data centers were serving the site. Today the average page contains 108 objects, requiring over 40 times more roundtrips to the servers, and thereby delaying full rendering of the page. Page size also increased significantly during these years, from an average of 14.1k to 1436k. And in spite of the improvements in bandwidth, this change significantly affected performance, increasing average load times by over 60%.

User Expectations Increase

As users continue to increase the time spent online, the statistics on their escalating expectations are sobering. In 2006, the average online shopper expected a web page to load in 4 seconds. Today's shopper expects a page to load in 2 seconds or less. Up to 57% of shoppers will abandon a site after waiting 3 seconds for a page to load, and 8 out of 10 people will not return to a site after a disappointing page load experience. Almost one third of these dissatisfied users will go on to tell others about their disappointing experience. Almost 60% of mobile web users expect sites to perform as quickly on handheld devices as they do on home computers, and about the same number of mobile users say they would be unlikely to visit a site again after a poor mobile web experience.

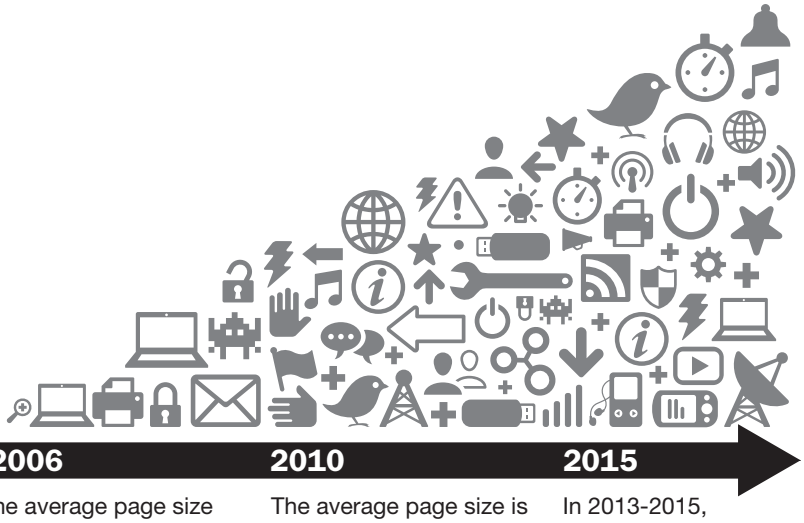


Figure 2: Web pages are bigger and more complex.

Applications Must Support Multiple Browsers

The days of being able to create websites that are optimized for only one or two browsers are clearly behind us. Browsers based on the WebKit open source engine - including Apple Safari and Google Chrome - as well as a growing array of mobile browsers, are increasingly taking market share from Internet Explorer and Firefox. Although Internet Explorer continues to be a widely used browser, its capabilities vary significantly across versions.

To be most effective for even a majority of users, optimization code must be tailored to the browser type and version used for each request.

Performance Tuning is Expensive and Never-Ending

Given the high demands that modern web applications place on servers, the high expectations of users, and the need to address a wide variety of browsers, performance tuning through code optimization becomes very expensive. As sites evolve, as usage patterns shift, as new best-practice coding techniques develop, and as new browsers and browser versions emerge, the task must continually be revisited, and the complexity of the task requires that it be assigned to senior-level programmers.

Alternative Approaches to Site Optimization

Because of the prohibitive cost of continuous optimization code maintenance, and because most organizations prefer allocating top programming resources to develop new features, the preferred route to improve performance is often to upgrade hardware or to engage external content delivery services.

57% of online consumers will abandon a site after waiting **3 seconds** for a page to load.

80% of these people **will not return.**

Of these, almost half will go on to tell others about their negative experience.

Source: PhoCus Wright

Data Center Build-Out

Adding and upgrading servers can improve site performance to a point, but the results are often disappointing because per-request server processing time represents only one small part of the performance matrix. Organizations are often disappointed to discover that adding or upgrading servers can have little effect on the time required to load each page.

To supplement infrastructure upgrades, U.S. companies spend an additional $4.2 billion per year on application delivery controllers (ADC) and content distribution networks (CDN).

ADCs and CDNs

An ADC is a type of network device that provides a set of processing features to optimize enterprise application environments. ADC devices are situated inside the data center firewall and represent an evolution from basic load balancers. In addition to load balancing, modern ADCs monitor server health, implement advanced routing strategies, and offload common server tasks such as SSL termination and TCP connection management.

A CDN is a service that provides multiple content caches that are distributed geographically across a service area. CDNs accelerate site performance and improve scalability by increasing the number of servers that fulfill requests for a site's content and shorten the physical distance traversed by those requests. In addition to the edge servers, CDNs also may provide redundant core and backup servers and redirection services that ensure high availability. They do not reduce the number of requests required to render each page and have no effect on browser efficiency.

Radware's Approaches to Site Optimization

Radware's approach to site acceleration is intended to complement, not necessarily replace other forms of site optimization, such as server build-out, ADCs and CDNs. Controlled performance studies have consistently shown that adding Radware's FastView improves performance, even when other solutions are already in place, by doing the following:

- consolidating and caching resources to decrease the number of roundtrips
- improving the utilization of browser caches
- reordering page operations to avoid blocks to rendering
- managing browser connections

Total Savings
FastView works in conjunction with content delivery networks and application delivery controllers to deliver cumulative performance benefits.

Unaccelerated site
Number of requests = 63
Payload = 769 kb
Page load time = 9.5s

CDN + ADC only
Number of requests = 63
Payload = 769 kb
Page load time = 6.2s

FastView + CDN + ADC
Number of requests = 9
Payload = 456 kb
Page load time = 2.8s

FastView Optimization Treatments

Radware's FastView is built on an extensible software framework that enables the company to deploy new or improved treatments. Radware continuously confers with industry leaders and performs its own research to ensure that its treatments continue to represent best practices and keep pace with browser innovations. The following sections drill down the technical details of several of the current set of treatments.

Resource Consolidation and Caching

Radware's FastView implements multiple treatments for consolidating and caching resources such as JavaScript, cascading style sheets (CSS), and images. These treatments reduce the number of roundtrips required to render each page and thereby speed up page loading. The system accomplishes this acceleration by consolidating resources of the same type into single resources and replacing multiple resource links in each page with a single link to the new consolidated resource.

Overview of Radware's FastView Web Performance Optimization

In addition to reducing the time it takes for each resource to be delivered from servers to clients, Radware's FastView focuses on reducing the number of requests required to render each page, reducing the number of bytes in page responses, optimizing the order in which requests are made, and improving browser rendering efficiency. Accomplishing this is a two-step process:

Record and analyze site traffic

Using an offline, asynchronous process that does not interfere with normal site operations, FastView samples all request/response pairs and dissects the composition of the site's pages. Employing proprietary heuristics, FastView detects patterns of resource usage and formulates rules for resource consolidation and caching, grouping resources according to the pages that request them. The system also tracks the rate at which resources change, classifying them as relatively static or relatively volatile resources.

This analysis process results in a dynamic cache of per-request processing instructions and resources. The cache is dynamic because the sampling process never ends. As new information about site usage is analyzed, cached instructions and resources are continually updated.

Intercept and modify site responses in real time

These modifications, which are highly optimized to occur very rapidly, change the behavior of the web pages returned by the site, without altering their appearance to the user. These modifications consist of a series of treatments that the system applies according to the cached processing instructions for each page, and these page modifications vary based on the browser being used for each request.

This two-part process is illustrated in the following figure. The block labeled "Response Analysis and Consolidation" represents the offline asynchronous process of analyzing pages, formulating rules and gathering cached resources. The block labeled "HTML Response Rewriter" represents the real-time process that intercepts pages and applies customized processing instructions to improve rendering performance.

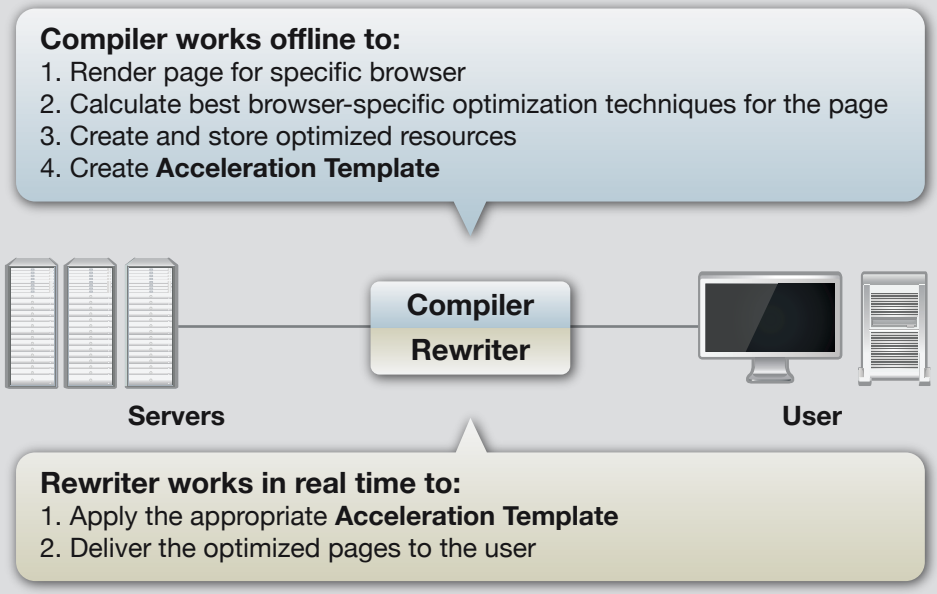


Figure 3: Radware's FastView web performance optimization consists of recording and analyzing site traffic and intercepting and modifying site responses in real time.

The offline analysis engine examines the site's page responses, sorting out which resources are used together, and developing rules for consolidating the resources. After sampling quotas have been met, the system creates and caches consolidated resource files based on the analysis findings. Then the page rewriter engine is ready to begin modifying pages as they are sent to clients, replacing multiple calls for individual resources with much fewer calls for the new consolidated resources. When necessary, the rewriter also adds code to the page that allows it to parse out the individual resources after they are delivered in consolidated form. The system then continues to examine and analyze the site's responses and adjusts its consolidation rules as necessary.

Many web pages contain a combination of static and dynamic content. For example, on the user profile page of a social networking site, most of the images, JavaScript and CSS are static, but specific user content such as the profile image or other user-added images often change. These resources will be different per user, and may even change for the same user over the course of a session.

One essential feature of Radware's FastView is to detect this dynamic data by sampling multiple responses and to recognize that it should not be consolidated. Even after consolidation, FastView remains vigilant in sampling unoptimized responses and invalidates or adjusts consolidation rules to ensure that dynamic data is passed through to the user. Configuration settings are also available to allow developers or administrators to flag certain content as exempt from consolidation treatments.

JavaScript Consolidation

One of FastView's resource consolidation treatments focuses on JavaScript. The JavaScript code used by a web page is often split into multiple files, either because developers use external libraries or because they create multiple files to implement their own code modularity. JavaScript can also be included from external links that dynamically create the script based on the link's URL.

FastView analyzes the JavaScript resources used in each page and determines whether or not they are static or dynamic. Once it has determined which resources can be consolidated, it then examines their contents to extract any inline scripts. Inline scripts include, for example, code that executes immediately when a page loads rather than in response to an event such as a body onload() event. The order and placement of each inline script is maintained during HTML rewriting, to ensure that the resulting JavaScript works correctly in the browser. The inline scripts are extracted and a processing instruction is created to rewrite the original script tags.

The remainder of JavaScript code, which generally contains only methods, variable declarations, constant assignments, or classes, is consolidated into a single JavaScript file. FastView creates a rule to insert a script link referencing this file right before the first JavaScript tag in the response. Another rule removes the script tags for the original JavaScript resources that were consolidated. In addition, the consolidated JavaScript is "minified" by removing whitespace and comments to further reduce the size of the resource.

CSS Consolidation

CSS consolidation is similar to JavaScript consolidation, but is not as complex. CSS does not execute inline, so after all of the CSS has been consolidated, a rule is added to insert a link to the consolidated CSS resource into the <head> section of the HTML response. By inserting the link in the <head> section, FastView ensures that the page does not have to wait for CSS while rendering, thus improving load times. Rules are also generated to remove links to the original unconsolidated CSS resources.

As with JavaScript, the consolidated CSS resource can also be minimized by removed whitespace and comments, as well as by replacing verbose CSS style syntax with equivalent syntax that is more concise.

Image Consolidation

Image consolidation is achieved by automating the implementation of a CSS technique called “spriting.” CSS sprites group multiple images into one composite image and display them using CSS background positioning. FastView brings the advantages of this advanced technique to applications that were not originally developed to employ it. Consolidating many images into a single CSS sprite significantly impacts performance by replacing multiple calls for the individual images with just one call for the sprite.

To implement spriting, FastView locates <img> tags in the HTML response, as well as any styles in the CSS that use background images. The system analyzes these styles to determine whether they can be supported by using a sprite. For example, a background image that has no fixed height or width but is positioned with the “bottom” or “right” keywords cannot be supported within a sprite. In this case, the optimizer recognizes that there is no way to use absolute positioning with that image and excludes it from the sprite. Most images, however, can be included.

Next, FastView determines how many sprites to create and what types. In general, the system generates at least one sprite per image type used on the page. For example, it creates one sprite for jpg images, one for gif images, and one for png images.

Further analysis ensures that the resulting rules produce optimized pages that are indistinguishable from the original unoptimized ones. For example, styles that use “repeat-x” and “repeat-y” often cannot be combined within the same sprite, because repeating images will overlap during sprite creation. If the original style uses a background image with a repeat-x or repeat-y, which usually is implemented with a 1px width, then the image is extended all the way across or down the sprite, so that when the repeating takes place it does not repeat the transparent space to the right or bottom of the image. If the original style positions a background image with a 100% width or height, then the image is right or bottom aligned in the sprite to achieve consistency with the original.

Sets of images that are used on multiple pages may also be consolidated into sprites. This promotes efficient use of the browser cache, because the sprite containing the consolidated images is downloaded once for the set and not once for each page that uses any of the images in the set.

When FastView rewrites the HTML of each page, it creates a new CSS class for each original image, using the correct sprite and positioning to duplicate the original page’s appearance. A rule replaces the “src” of the original image with a small transparent image (1x1px) and modifies any CSS class for the image to use the new class. Similarly, existing background image styles are modified to use the new sprites. FastView modifies the background-url attribute to reference the correct sprite, and sets the position-x and position-y attributes to the positions of the offset location of the image in the sprite. If position-x and position-y are already in use, the offset is added to the original values to ensure correct positioning.

It is apparent how difficult and time consuming it would be to apply these techniques manually to every page in a web site, and how much work would be involved in maintaining the code as the site changed. This is especially true because the code must be customized for each type of browser. Radware’s FastView is like a coding robot that continuously performs these tasks automatically, rewriting code on the fly very rapidly as pages are delivered to users, based on rules and cached resources that are constantly updated based on offline analysis.

Image Compression

Organizations are using graphically reach designs to make websites more user friendly and appealing to the eye. However, this comes with a price, and images are taking more than 55% of the average web page’s payload. Yet most sites fail to optimize images, or use image optimization techniques like Progressive JPEG which accelerate the image initial download but worsen the overall user experience.

PerfectImage is an algorithm developed at Radware to optimize images - significantly reducing image file size while ensuring minimal image quality degradation. By using Digital Camera Images (DCIM) human vision algorithm, images are carefully compressed using various image compression formats (such as JPEG, JPEG 2000, WebP, and others) so that the final image has less than 1.5% difference vs. the original image, which is invisible to the human eye. As a result, the image file size can shrink by up to 80%.

The PerfectImage algorithm, incorporated in FastView, optimizes images embedded in web pages on the fly. For every browser request, PerfectImage selects the most efficient image compression format supported by the browser, and that yields the highest compression ratio. That way, image compression is optimized per request, per browser type and version, while ensuring quality degradation is minimized and remains invisible to the human eye.

Chunked HeadStart

The preceding discussion of CSS and JavaScript consolidation describes techniques that move certain segments of code up into the <head> section of a page to avoid having them block rendering activities that occur in the body of the page. One further extension of this basic technique is a proprietary Radware treatment called “Chunked HeadStart.”

The name of this treatment derives from its use of HTTP’s chunked encoding, which allows servers to begin responding to requests before the entire response is available to be served. In addition to using the <head> section of a page, as described above, this treatment also makes use of the fact that browsers are capable of handling two separate <html> sections for a single page. FastView can implement rules in a new <html> section inserted above the page’s main <html> section, which can remain untouched by this treatment.

Once FastView has performed offline analysis of several requests and responses for a given page and has determined that the page would benefit from Chunked HeadStart, it formulates a rule that is applied as soon as a request for the page is received. FastView initiates chunked encoding and sends back to the client a partial response that includes links for the resources required for the page, often in consolidated form. FastView forwards the original request to the Web server so that the balance of the page can be constructed. While the server is completing the response, the browser is already processing the first part that was sent by FastView. The browser retrieves and caches the resources need for the page, so that when the remainder of the page arrives, the browser can retrieve the required resources from its cache.

Chunked HeadStart enables a form of distributed parallel processing, whereby resources are fetched and cached by the client while the server is still creating the response that will use these resources.

Dynamic Resource Discovery

As websites continuously work to enhance website functionality, they make increasing use of applets and programs that run on the browser side (such as Java Scripts and CSS templates). The execution of this code often causes the browser to dynamically call for additional resources that are required for proper presentation of the page, resources that are invisible to standard WPO engines and thus can’t be optimized.

To ensure FastView can apply its various optimization treatments on all objects, even those dynamically called by code running on the browser, a new mechanism was incorporated into FastView, which allowed it to simulate the browser behavior and execute the code running in that page. As a result, FastView dynamically discovers all objects required by that web page, and apply its various optimization treatments on all of these objects. As an example - many of the images are called by CSS templates (e.g. background images) and by Java Scripts. Combining FastView’s dynamic discovery capabilities with PerfectImage, advanced caching, preloading and other acceleration treatments, FastView can apply more optimization treatments on more objects per page and accelerate web pages even more.

Optimizing Browser Caching

Chunked HeadStart is just one of several FastView treatments that optimize browser caching. The most efficient way for a page to complete a resource request while it is rendering is to obtain the resource from the local browser cache. In addition to using Chunked HeadStart to populate the browser cache with resources needed for the current request, FastView also seeds the browser cache with resources that are likely to be needed for subsequent requests.

Part of the offline analysis that FastView performs is to detect navigation patterns. FastView learns which pages are most likely to be requested following a request for a given page. Based on this analysis, the system creates and applies rules that send these anticipated resources to the browser after rendering of the current page is completed. When the anticipated future requests occur, the required resources are already cached on the browser and no further roundtrips to the server is needed, significantly speeding up the rendering of those pages. The rules used for this predictive browser caching are continually reviewed and updated by FastView, based on the latest navigation statistics. In addition, the system carefully sets and adjusts invalidation rules for cached content to ensure that users aren't exposed to out-of-date versions of resources.

Another treatment for optimizing browser caching relies on an analysis algorithm that distinguishes a user's initial visit to a page from repeat visits to that page. Based on usage statistics, FastView identifies pages that are likely to be "landing pages" that are often the first page of a site that a user visits. The system then employs several mechanisms to distinguish when identified landing pages are actually being revisited rather than visited for the first time.

When a page is being visited for the first time, no resources for the page are yet present in the local browser cache, and the most efficient way to deliver the page is to embed all required resources within the initial page response. However, once a page has already been visited and the browser cache has been populated with its resources, or once visits to other pages on the site have pre-populated the browser cache with resources for the page, then it would be inefficient to send all resources inline. For this reason, performance is significantly improved by the system's ability to detect when a page is being visited initially as a landing page and when the request represents a repeat visit to the page.

Payload Reduction

In addition to reducing roundtrips to the server, FastView also reduces the size, or "payload," of each response. In some cases, this involves standard compression techniques, such as image compression and the use of gzip. As previously noted, FastView also "minifies" CSS and JavaScript by eliminating white space and comments, and by replacing verbose syntax with more concise syntax that provides equivalent functionality.

Another approach to payload reduction is to extract large chunks of page content that are not actually needed on the client. For example, ASP.NET pages often contain a large hidden control that stores page state. This View State content is not needed to render the page. Instead it is used by ASP. NET to detect user entries, actions, or modifications when HTML forms are posted back to the server.

FastView performs a View State-specific treatment that extracts this data from responses, tags it with a unique key, and stores it in the memory store of the optimizer engine. The unique key is inserted into the page, and when a future request posts back the page, the system uses the unique key to retrieve the stored ViewState data and re-insert it into the request.

Connection Management

Although modern browsers are capable of opening more than one TCP connection to a domain, they still are rarely able to determine the optimum number of connections to open. FastView implements a treatment that selectively "tricks" the browser into opening additional connections by using prefixes to create varied host domains. This technique, sometimes called "domain sharding," makes the browser think that it is communicating with multiple servers and therefore should open additional connections.

For example, some resource links to www. domain-name.com are rewritten as 1.www. domain-name.com or 2.www.domain-name. com. The browser will then open additional connections for each of these new host names, increasing the number of parallel downloads from the site. Implementation of this treatment also requires

the creation of DNS entries to direct all requests to the original IP address. Of course, too many concurrent connections can degrade performance rather than enhance it, so FastView carefully manages this treatment to ensure optimum bandwidth utilization for each type and version of browser.

Deployment Options

FastView provides multiple deployment options:

- Appliance – a hardware network appliance manufactured by Radware and installed within the company's network.
- Virtual – a virtual machine that is installed as software on the company's server.
- Cloud Service – Web traffic is directed through a virtual FastView instance that is external to the company's data center.

These configuration options are illustrated in the following figure.

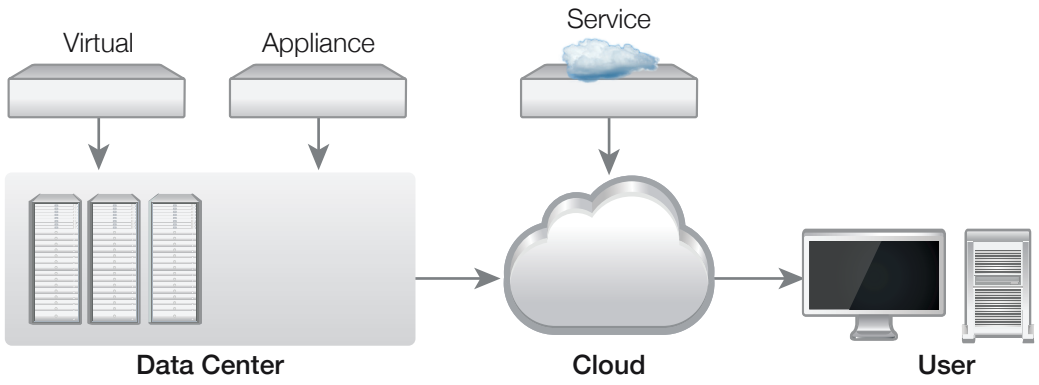


Figure 4: Multiple deployment options.

Fastview Appliance

When implemented as a hardware appliance, Radware's FastView supports both vertical and horizontal scalability through supplemental hardware installation. In the rare case of an unresolved device failure, traffic automatically routes around the device, allowing unoptimized responses to be served until the system is restored.

Fastview Service

Radware's FastView Service uses DNS settings to redirect all web traffic to pass through a virtual FastView instance that is external to the company data center, in "the cloud". Because of this, it can be implemented quickly – with one flip of a switch – and scaled rapidly. The FastView service also supports applications in physical and virtual environments and can easily be scaled and deployed on demand.

The FastView service is identical to the appliance in that it applies the same optimization technologies and treatments, has been subjected to the same rigorous internal testing, and delivers identical acceleration results.

Fastview Virtual Appliance

The FastView Virtual Appliance (VA) is designed to run over VMware Virtual Infrastructure. It has been engineered to meet key virtualization requirements, ensuring its interoperability with virtualization tools. With the exception of how it is delivered, the FastView VA is identical to the FastView appliance. Like the appliance, it is available in three versions: FastView 600, 1800 and 3600.

Platform	Supported	os version
VMware Virtual Infrastructure	Yes	ESX/ESXi 3.0 (3.0.0+)
VMware vSphere 4	Yes	ESX/ESXi 4.0 (4.0.0+)

In any deployment – appliance, service, and virtual – FastView can complement a load balancer or ADC deployment, significantly enhancing its acceleration functionality and offloading efficiency. High availability device configuration to provide automatic failover.

Conclusions

Companies that depend on web-based applications to generate revenue, soon realize that site performance directly correlates with increased conversions, increased shopping cart size for ecommerce, increased page views for advertising, and higher search rankings. Yet consistent high performance is very difficult to ensure, even with expensive hardware and service additions, because none of these additions can compensate for the inevitable inefficiencies in page construction and rendering.

Radware's FastView is unique in its ability to address application-level inefficiencies that could otherwise only be solved with continuous recoding by a devoted team of skilled programmers. Using an extensible and self-tuning set of proprietary algorithms, FastView automatically and safely improves page performance without altering the final appearance of the page.

FastView does not require software, server or network changes, and it can be implemented easily in a variety of hardware-based, software-based, or service-based configurations, yielding immediate verifiable results. Any previous or subsequent investments in ADC or CDN devices and services are complemented by the traffic reduction and enhanced browser caching that FastView enables.