The Increasing Complexity of Web Traffic

The Web is widely used as the front-end for a range of services. Currently the Web is shifting from static Web pages to dynamic interactive Web applications that use JavaScript, and the impact of this is showing up in Web traffic. Here we examine Web traffic behavior at the HTTP request level, and discuss the complexity of bottlenecks arising from JavaScript processing. In closing, we summarize the future outlook for the Web.

3.1 The State of the Web

The Web began as unsophisticated Web pages written as simple HTML files, but now Web applications have come to be used as the front-end for a variety of services. Services that have been used with dedicated applications on the desktop until now are also appearing as Web applications one after another. The methods and scope for use of Web applications continue to grow. In addition to reading news and blogs, users are accessing Web applications via interactive UI (user interfaces), such as for search, email, social network services, and maps.

Web application services are becoming more multifaceted, and user environments are also diversifying. It is likely that some readers of this article use a laptop or desktop computer at work, while using a smartphone or tablet during their commute or at home. Along with changes in user devices, network environments have also changed from wired connections to mobile connections such as 3G and LTE, as well as Wi-Fi.

As the range of Web applications and user environments become more diverse, in recent years Web traffic volume has been increasing again. One reason for this increase is that content formerly not handled using HTTP is now delivered over HTTP. For example, videos that were delivered using other protocols such as RTSP in the past are now delivered via Flash, making it possible to provide them from Web servers using the HTTP protocol. It has been reported that the availability of download services for large-volume content via Web applications is another reason that Web traffic volume is increasing. This demonstrates that user demands on Web applications are mounting as a larger variety of Web applications are used, and overall Web traffic volume increases.

There are a number of metrics for comparing the performance of Web applications, but one of the most important is response time (which we will define here as the time taken until display is complete). A number of research papers and reports identify that having a fast response time is extremely important for Web applications. For example, reports indicate that for Microsoft's Bing or Google Search, even a delay of a few hundred milliseconds in the speed of displaying search results can lead to decreased revenue and lower subsequent visitor numbers. It has also been said there is a three second rule for the time it takes to complete the display of a Web page, but in certain online shopping statistics it was reported that users actually expect content to display in less than two seconds.

To meet the high requirements of users, such as having a sophisticated UI and a fast response time, and to provide a variety of services, Web applications have become more complex. As a result, recent Web traffic has also become extremely complex in comparison to its simple beginnings, when HTTP GET was used on HTML. Next we will examine the kind of Web traffic actually generated behind the scenes by recent Web applications.

3.2 Recent Web Traffic Characteristics

To find out the status of recent Web traffic, we will first look at Web traffic for views of the top page of Yahoo.com, which is an example of modern Web applications. Figure 1 shows the top page of Yahoo.com on November 5, 2013. Let us take a quick look at the figures for Web traffic generated when this top page is accessed.

HTTP requests: 83
Amount of data transferred: 978.7 KB
Number of domains connected to: 13
Number of TCP connections: 39

These results are merely an example of viewing the top page of Yahoo.com on a given day using a PC browser (Firefox), so these values will vary depending on the user environment and content. However, comparing them to Figure 1, I believe you get a sense of the Web traffic volume that we aren't usually aware of.

You may be wondering whether these Web traffic figures for the top page of Yahoo.com are smaller or larger compared to typical Web applications. To answer this, we compared them with figures provided by the HTTP Archive*1. The HTTP Archive obtains Web pages from major Web applications based on Alexa*2 and the Fortune 500 every two weeks, and automatically analyzes it to produce reports. The average Web application on the HTTP Archive generated over 90 HTTP requests per page, transferred over 1,400 KB of data, and connected to over 15 different domains (yearly averages for 2013). This indicates that the average Web application generates more Web traffic per page than we saw in the example for the top page of Yahoo.com.

Now that we have identified recent Web traffic volumes, we will next take a look at the kinds of content included in this Web traffic. Figure 2 shows the content included in the top page in Figure 1 classified by content type. The pie chart on the left compares the number of pieces of content, and the pie chart on the right compares content sizes. We classified the types of content as HTML, JavaScript (JS), Cascading Style Sheets (CSS), images such as JPEG/PNG/GIF (Images), and Other. Figure 2 shows that for both the number of pieces of content and content size, images accounted for the largest ratios at over 70%, with JavaScript second at over 10%.

Figure 1: Top Page of Yahoo.com

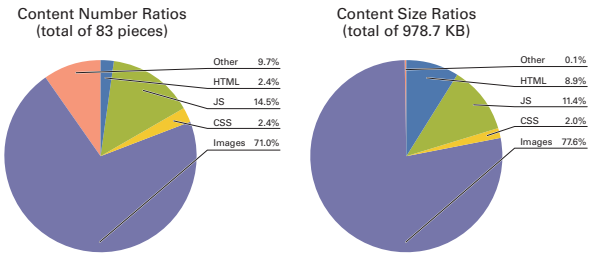

Figure 2: Content Type Ratios

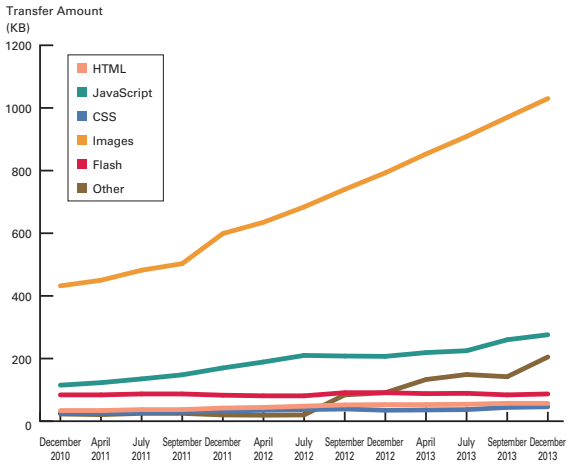

Figure 3: Content Type Transfer Volume Trends

*1 "HTTP Archive" (http://httparchive.org/index.php).
*2 "Alexa" (http://www.alexa.com).

Let us examine how these content type ratios compare with typical Web applications. Figure 3 shows average content type ratios (transfer volume) for Web applications over a period of three years starting from December 2010, using the HTTP Archive data mentioned above. We can see that over these three years, transfer volumes for images and JavaScript have more than doubled. It is also evident that, like the results in Figure 2, images accounted for the largest ratios in Figure 3, and JavaScript had the second largest ratios.

Looking at Figure 1, we can see that images are used extensively in a variety of places, such as thumbnail images and icons. However, it is less clear where JavaScript, the second most common element, is used. For the Figure 1 example, given that this is the top page of Yahoo.com, the HTML file indicates that many YUI (Yahoo! User Interface Library)*3 JavaScript libraries are accessed. These JavaScript libraries are used to implement sophisticated UI elements, such as switching the images displayed for the main topics at the top center of the page in relation to the mouse's position, and automatically switching images. Although not directly related to visual elements such as UI, JavaScript is also used to obtain ad click and user flow data.

This demonstrates that the relative importance of JavaScript use in Web applications has been increasing in recent years. JavaScript began to attract attention from around when Google's map service was launched in 2005, and about the same time that the word Ajax became common.

3.3 Accelerating Response Speed and Increasing UI Sophistication Using Ajax

Before Ajax (Asynchronous JavaScript + XML) appeared, Web applications did not send requests from browsers to Web servers until user input was determined. The application server and database processing required for responses began after the Web server received a request from the browser. This meant that browsers could not begin displaying content until server-side processing was completed and the response was received from the Web server. Additionally, because the display of content also involved redrawing the entire Web page (screen transition), depending on the type of Web application, obtaining data and processing the drawing of the entire Web page could take time.

The advent of Ajax enabled communications to be carried out between the browser and Web server even while users were performing actions. This made it possible for Web applications to conduct background processing server-side in conjunction with the application server or database based on details the user was still inputting but hadn't finalized, such as mouse movement or keyboard input. It also enabled browsers to redraw parts of a Web page based on responses from the server, making it possible to reduce the time taken to obtain Web page data, and lessen the drawing processing load.

For achieving real-time response speed and an interactive operational feel in Web applications using Ajax, it is crucial to have a system for asynchronous communication between browsers and Web servers, as well as effective background processing for each Web application. Next we will examine the kinds of requests and responses actually sent and received by services via Ajax, using search suggest functions and map services as examples.

Suggest functions display a list of frequently searched for search terms that start with the same letters of a partially entered phrase. Figure 4 shows the suggest function results when searching for the term “IIJ” using Yahoo Search. We can see that different autocomplete results are displayed as each letter of “IIJ” is input. By their very nature, it is expected that suggest functions will display autocomplete results quickly. To understand how this system displays

Figure 4: Search Field Input and Suggest Function Examples

*3 “YUI” (http://yuilibrary.com).

autocomplete results quickly, we will take a look at the HTTP requests and responses when suggest functions are used. As shown in Figure 5, when "I" is input into the search term input field, the browser sends an HTTP request to the Web server that includes a "p=I" query in the URL like the one displayed at the top of the figure. In response to this request, the Web server returns the data for the autocomplete results to display via the suggest function as JSONP format data. The browser redraws only the autocomplete results part for the search term by processing this data using JavaScript obtained in advance from the Web server. In the example of the first letter from Figure 4, as shown in Figure 5, data such as "イオン", "ipad" and "一休" that matched the lower-case "i" was returned to the browser (results from November 4, 2013). This demonstrates that search term suggest functions make it possible to display suggestion results quickly by enabling minimal HTTP requests and responses for exchanging only the necessary data, and partially redrawing the browser screen using the data received.

In map services using Ajax, the map images move around following the mouse position. Like the suggest function we examined above, the browser only redraws part of the screen for maps as well, rather than refreshing the entire page. To achieve this, the browser detects mouse movement when using a map service, and requests the missing parts of the map from the Web server. In response to the browser's request, the Web server sends the missing map data to the client, and the client corrects the data received and displays the required parts. However, unlike the suggest function previously mentioned, with map services complex map data is sent from the server to the client. This means that compared to the small amount of data used for suggest functions, a larger amount of data is sent from server to client. In light of this, prefetching is used as an important component technology. This predicts locations that may be displayed next in advance, and obtains the map data ahead of time to shorten the time that users are waiting for data to download. Figure 6 shows the results displayed when a search for Jimbo-cho is input in the Yahoo Maps search field. The mouse has not been moved after the results were displayed, but looking at the requested map data, data for parts not displayed in the browser continue to be obtained after acquisition of the data displayed in the browser is complete. For the example in Figure 6, data for a number of other parts not displayed in the map was obtained in advance, such as the area south of the displayed map, and data for when the display area is zoomed out. Obtaining and displaying only parts of the map that are missing, and predicting the user's next action to obtain data in advance while they view the map, enables a smooth operational feel for map services.

3.4 The Complexity of Web Application Bottlenecks

When Ajax first began to be used, it required JavaScript code tailored to the different browser implementations to be prepared on the Web application side. However, jQuery*4, prototype.js*5, and Google Web Toolkit have recently been made available as application frameworks, making it comparatively easy to incorporate Ajax into Web applications. While using JavaScript libraries in Web applications enables fast, user-friendly UI to be provided to users, dependencies between JavaScript files and for the execution sequence of drawing processing have made it more difficult to find Web application bottlenecks.

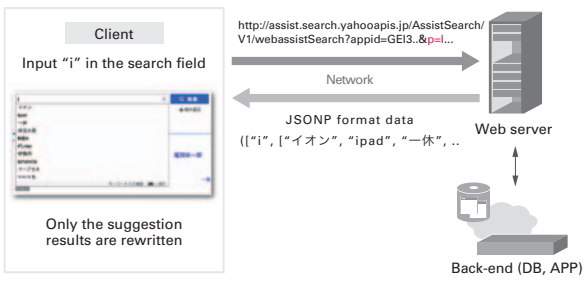

Figure 5: Suggest Function Requests/Responses

Figure 6: Prefetch Example for Map Services

*4 "jQuery" (http://jquery.com).
*5 "Prototype.js" (http://prototypejs.org).

During the drawing process, browsers generate intermediate data that they can interpret directly called object models (DOM and CSSOM) from the HTML and CSS. If a Web page does not contain JavaScript, a render tree is created from the DOM and CSSOM, the display layout is determined, and actual drawing is carried out. When JavaScript is added on top of the HTML and CSS, the drawing processing flow becomes intertwined due to three-way dependencies. Because JavaScript requires the content layout to execute, it waits until the CSSOM is created from the CSS that manages the layout before executing. JavaScript execution must also be completed before creating the DOM from HTML, so the browser waits for JavaScript execution to finish before executing DOM creation.

Let us look at an example of the HTTP requests when the IIJ website is viewed to see how dependencies in the execution order between content like this actually affect the execution of Web applications. Figure 7 shows a network waterfall for HTTP requests when the IIJ website is specified in WEBPAGETEST*6.

WEBPAGETEST enables you to examine Web application performance by specifying the URL and geographic position to check, the browser type, and the network bandwidth and RTT. This time we implemented the test by specifying Mobile 3G-Fast (1.6 Mbps 150 ms RTT) for network bandwidth to make the impact of content dependencies clear.

Network waterfalls display a top-to-bottom list of HTTP requests in the order they were sent. They can also show information such as whether or not DNS lookup is performed for each HTTP request, the time of lookup, whether or not 3-way handshaking is carried out for TCP connections, and the time that the requested content is downloaded. This makes them suitable for investigating the behavior of individual HTTP requests.

Let us take a look at jquery.js, the sixth from the top in Figure 7. This content took approximately two seconds to download, which was much longer than other content. This jquery.js is the core file for jQuery, which was previously identified as an application framework. After the sixth item obtained, jquery.js, there are a number of items from the seventh onward that also involve the download of JavaScript files related to jQuery. Of these JavaScript files related to jQuery, jquery.js is the largest at 239 KB, and as a result it takes longer to download.

Drawing a vertical red line on Figure 7 at the point where the download of jquery.js completes, it lines up with the start time for the download of the content group requested from item 20 onward. Checking this against the drawing processing order for HTML, CSS, and JavaScript mentioned above, the browser waits for the execution of JavaScript including jQuery until the download of jquery.js is completed. During this time, HTML parsing, DOM creation, and the download of external content embedded in the HTML are all suspended. Once the download of jquery.js and the execution of JavaScript was complete, we believe the download of external content such as images from item 20 and beyond began all at once due to progress in DOM creation.

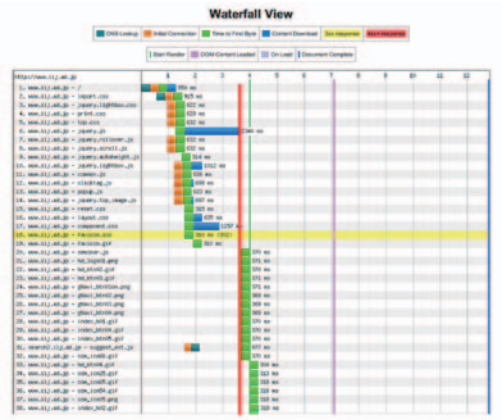

Figure 7: WEBPAGETEST Waterfall View

*6 "WEBPAGETEST" (http://www.webpagetest.org).

Here we have looked at comparatively easy-to-understand examples of execution order dependencies between content, as well as their scope of impact. Examples in which the processing of other content is blocked due to JavaScript execution, such as those given here, have also been found in many other Web applications, and the dramatic impact this has on Web application response time has been identified. However, when content has a hierarchical structure with multiple layers, or when there are convoluted dependencies between a number of pieces of content, it is difficult to identify the content or dependencies causing bottlenecks, as well as the scope of impact for these bottlenecks, from a network waterfall alone. Additionally, because user environments such as execution environments and network environments are becoming more diverse, it is also necessary to take into account differences in the degree of impact that bottlenecks have in a number of usage environments. This is making it even more difficult to identify bottlenecks.

3.5 Future Consolidation and Simplification of the Web

Here we have mainly looked at Web application behavior related to JavaScript, but JavaScript is only one of the component technologies used in Web applications. A variety of other technologies are also being used, such as the new HTML5 and CSS3 standards introduced to improve the performance of Web applications. There is also the collection of user information using cookies, etc., to personalize content such as targeted ads, and coordination (mash-ups) between Web applications using external Web APIs.

Content-based consolidation technology for Web applications is also being used to speed up the response time of Web applications, such as CSS sprites that display only images in a certain location from a number of images organized together. Another example is JavaScript minification, which involves the elimination of spaces and line breaks in JavaScript files. In addition to this consolidation of Web application processing by combining or applying existing technologies, there is protocol-based consolidation and acceleration such as the work on the development of HTTP 2.0 currently being carried out by the IETF*7.

As shown here, there are currently moves to consolidate some areas of Web applications, and it is expected that this will lead to the simplification of some aspects of them. On the other hand, it is also anticipated that consolidation will cause processing to become more complex.

As an ISP, understanding Web traffic behavior and its characteristics is extremely important for planning efficient operations. For this reason, we believe it will be crucial to continue to stay abreast of technology and other developments related to Web applications. We also think it will be necessary to visualize various aspects such as Web application structure and browser processing in addition to the network side of things, and we would like to continue developing technology for achieving this.

Author:

Megumi Ninomiya
Researcher, Research Laboratory, IIJ Innovation Institute. Ms. Ninomiya is engaged in the research of Web traffic.

*7 "IETF" (http://www.ietf.org).