WHITE PAPER

How Rich Internet Applications Impact the Network

www.crescendonetworks.com

Corporate Headquarters
6 Yoni Netanyahu Street
Or-Yehuda 60376, Israel
Phone: +972-3-538-5100

US Headquarters
633 Menlo Avenue, Suite 230
Menlo Park, CA 94025
Phone: (866)830-0400

Contents

Introduction

Popular web services such as Google Maps, Google Mail, and Yahoo's Flickr have brought Rich Internet Applications (RIAs) to the forefront of web technology. We hear terms like AJAX and Flex more often now and there seems to be a shift in the way web applications are designed and deployed. This paper provides a brief introduction to RIAs, what they are, how they're different from traditional web applications, and how they impact the underlying network. We will also discuss how Crescendo's AppBeat DC can help web applications as they migrate to these new deployment models.

RIAs: What Are They?

RIAs present an entirely new approach to web application design. Typical web applications have followed a simple submit-and-render model where the user looks at a page, fills out a form or follows a link, submits the request to the server, and the server returns an entirely new page back to the client. This process continues until the client session is done. Here, the browser is nothing more than a rendering engine. All application logic (e.g page format/layout, user information, etc) resides on the server side and the browser's only responsibility, more or less, is to display the formatted HTML pages sent to it by the server.

Although this model works, for a single user it typically presents minutes of application inactivity (while the user reads the page or fills out the form) followed by seconds of full-blown application processing (once the request is submitted to the server). RIAs strive to even things out a bit more, while at the same time tapping into the significant processing power of the client machine itself – a resource that's not currently being harnessed. And, of course, RIAs try to create a new, richer level of user experience. Simply put, RIAs take some of the application logic from the server and place it on the client. As such, they present a richer user interface resembling stand-alone desktop applications.

With an RIA, some of the logic is now within the browser itself, allowing the browser to actually process the application locally, rather than just provide a page rendering utility. Google Maps and Google Mail are perfect examples of such RIAs. These applications are much richer and more responsive than traditional web applications because the browser itself is now doing some of the work. At first glance, it may seem as though browsers would need enhancements or additional plug-ins to be able to participate in RIAs. However, popular browsers already have many of the capabilities RIAs need in order to operate.

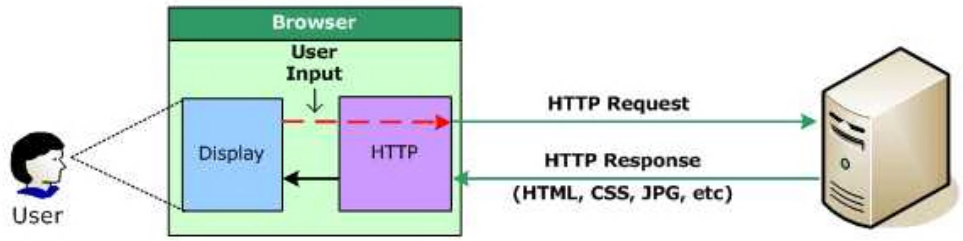
The most popular client-side capabilities used with RIAs are Javascript and Flash. Javascript is inherently supported by all popular browsers and Macromedia's Flash (now Adobe's) is a common enough plug-in that we'd be hard pressed to find a browser

without it. AJAX, which stands for Asynchronous Javascript and XML uses the browser-side Javascript facilities. AFLAX, which stands for Asynchronous Flash and XML uses the Flash plug-in capabilities of the browser. Since Javascript is natively supported by browsers, commercial implementations (like products from Backbase) simply provide an AJAX framework for development. Likewise, Flex is Adobe's framework for creating Flash-based RIAs. These are all terms that commonly refer to various RIA technologies and although they differ in functionality, they are all essentially serving the same purpose: creating an advanced web application that uses the client itself for processing some of the logic. RIAs are essentially creating a new paradigm for how web applications are being developed and deployed, a collective technology that is now commonly being referred to as "Web 2.0."

RIAs: How Do They Work?

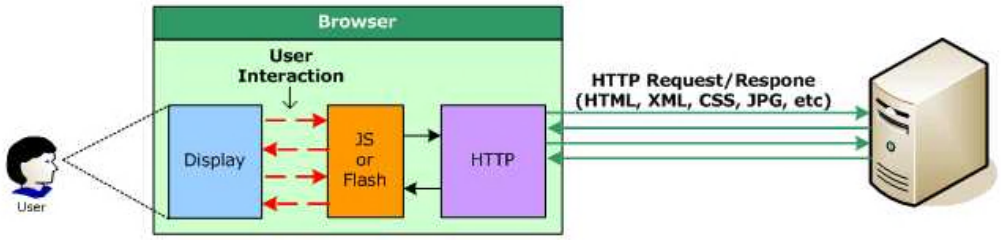
As discussed above, RIAs bring some of the logical processing of the application to the client browser, creating a richer user experience through interfaces that resemble traditional desktop applications. Google Mail, for example, is radically different than traditional web-based mail applications. Google Maps is another good example. The drag-and-pull functionality is very different than traditional map services, as is the fact that street layouts beyond the active window are already available if a user needs to look beyond the initial view window. Google uses AJAX for these services and these applications are just a few examples of the large number of RIAs that are popping up all over the Internet. But, how do these RIAs work? How is the information there before we click on it? And how are these applications different than traditional web apps? To answer these questions, let's look at a sample shopping site.

In a simple shopping application, a user first follows some links or submits a search in order to reach the desired item for sale. Each click of a link or submission of a search term sends a request through HTTP to the server. There, the server processes the request, generates a page and then returns it to the client. The browser then renders the page and displays it to the user. The exchange continues until the user finds the desired item. This process carries on as the user places items in a shopping cart, proceeds to check out, fills out payment/shipping information, and finishes the session. This is the approach of the traditional web application. Each user submission causes the browser to send a request to the server, where the server creates customized pages for this user and returns them to the browser. The browser acts as nothing more than a rendering engine, simply taking formatted pages from the server responses and displaying them properly to the user. The diagram below shows the basic framework under which traditional web applications have worked:

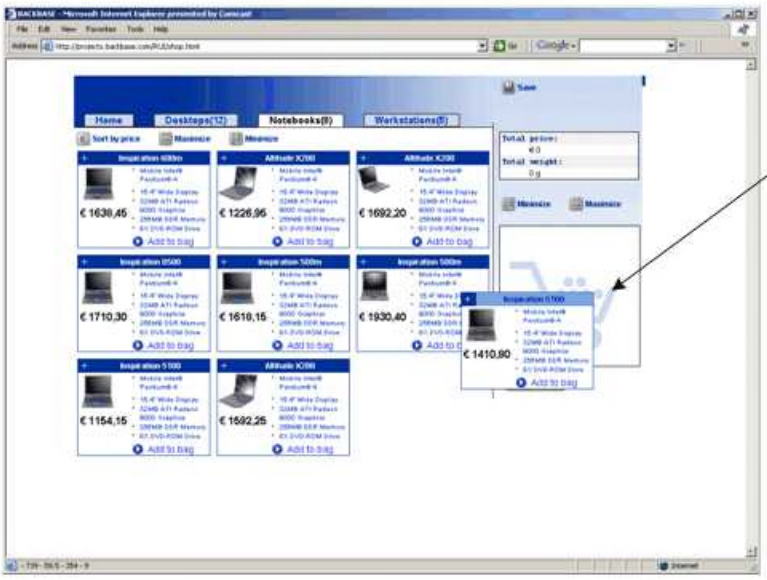


Although the model works, there are a number of inefficiencies in the way it operates. For example, with each user submission, the entire page is reconstructed by the server and sent to the user, even if the submitted information only changes a small portion of the page. This is true because HTML pages are seldom, if ever, compartmentalized where only sections can be refreshed while other portions remain intact. Likewise, while the user reads a page and/or fills out a form to submit, there is no activity between the browser and the server. This inactivity can last minutes.

Then, when the user is ready to move to the next page (by following a link or submitting a form) the server is given all the work to move to the next page at once, for processing. RIAs work very differently than this traditional model. The browser and server components are the same as before, and the communication still occurs over HTTP. However, there is an added client element that creates the rich user interface itself on the browser, providing a more interactive client experience. In contrast to the traditional web application model, an RIA can be visualized through the following diagram:



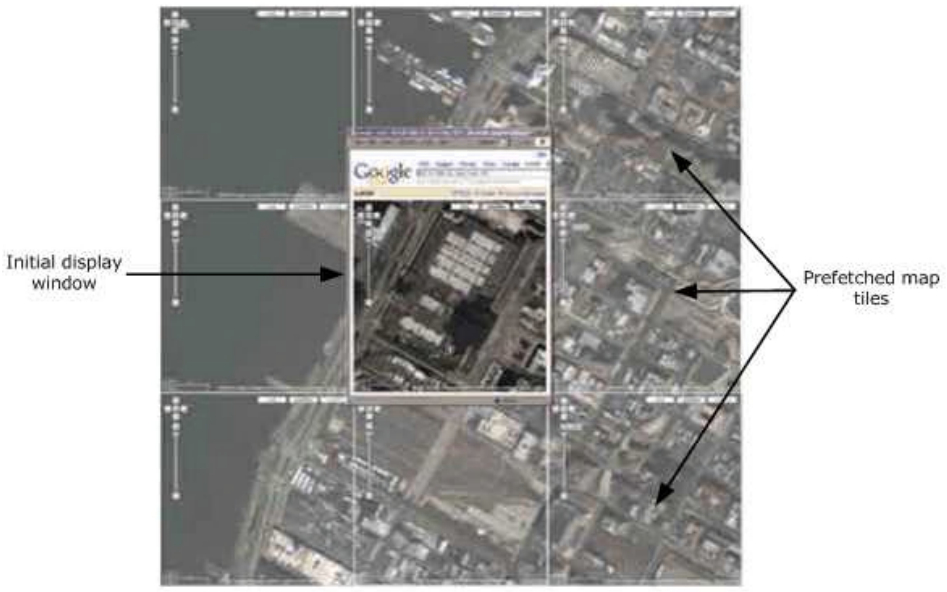
Here, there is an added element on the client browser (Javascript or Flash). The result is an environment on the user side that resembles a desktop application rather than a common web application. The user actually interacts with the browser-side part of the application locally, without necessarily exchanging any data with the server. For example, if our shopping site application used RIAs, items may be movable to the shopping cart using drag-and-drop. It's even possible that shopping carts could be created locally (without any messages necessarily being sent to the server) and then sent to the server whenever the application saw fit.



Essentially, RIAs take some of the application logic from the server and bring it to the client. No longer does every client click need to go all the way to the server and back - some of the stages of a transaction can happen on the browser. For example, tables can be sorted locally or full email threads can be viewed without having to wait for the server to reconstruct a conversation history. The communication between the browser and the server is no longer directly related to user interaction. What the user does and sees is independent of the way the application within the browser communicates to the server. They may or may not be closely tied, hence the asynchronous nature of RIAs. RIAs open a host of new doors for developers when creating rich, user-friendly web applications. Because of this new paradigm, there are a number of new concepts that become relevant in RIA development and merit discussion. Some of these concepts are ways developers can use the power of RIAs to enrich the user interface while others are new necessities that RIAs have to take into account.

Prefetching

Simply put, prefetching anticipates user patterns and proactively requests content from the application so that it's already available when the user actually requests it. Google Maps is a perfect example of prefetching in an RIA. When a location is mapped, the user can drag and scroll outside the immediate display window to see surrounding neighborhoods without any application delay. This happens since the tiles outside the initial display window were prefetched in case the user wandered there.

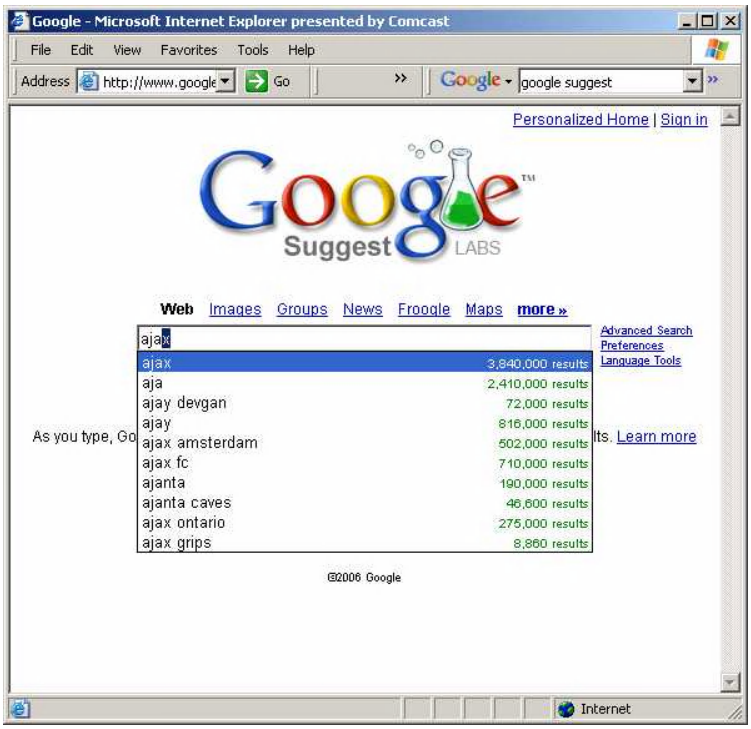


State Synchronization

In traditional web applications, all application logic (including session state, user information, etc) resides on the server side. With RIAs, since some of the application logic has moved to the client, there needs to be a scheme where both sides of the application (server-side and client-side) are synchronized about the state of the user session. Because of this, extra communication between the client and the server may be necessary. Albeit in the background and invisible to the client, this communication is essential in maintaining a consistent state between all application components.

Real-time Updates

Since RIAs create a more intelligent user-side element to applications, they may also offer real-time updates for the application. For example, Meebo (an AJAX-based instant message emulator) uses the client side of the application to periodically poll the server to check for new messages. The Google Suggest service is another prime example of real-time updates. This service extends Google's standard search functionality and tries to "guess" what the user is looking for by suggesting a list of popular search terms with every new letter typed into the search window. This means that with every keystroke, the client-side part of the application sends a request to the server to adjust the search suggestions.



Any other real time application will also need to have this back-and-forth between client and server. Updates can be done by either "pulling" data from the server (active polling by the browser) or by the server "pushing" the updates to the client. Either way, the now more intelligent client-side application needs to be constantly updated if the application requires a real- time interface. These are just samples of new concepts that come into play when designing and deploying RIAs. These issues are new to web applications due to the nature of RIAs and the way they operate. The basic lesson here is that although RIAs are a dramatic shift in the way web applications are created in usability and interface dynamics, they involve new issues to consider – especially when evaluating the impact of RIAs on the underlying network.

The Impact of RIAs on the Network

When it comes to deploying RIAs, network administrators must be aware of the new traffic profiles imposed by the application since it may affect their infrastructure and the way the network and its servers need to be provisioned. The concepts discussed above, along with other new facets of RIAs, pose new issues for network administrators to take into account. The intelligence on the client side often leads to an increase in the amount of traffic between the client and the server. Here, we will discuss some of the impact points that RIAs may impose on a network and server infrastructure. One or more of the following issues may arise in an application deployment, depending in the RIA.

- **More TCP Connections -** RIAs use the networking characteristics of the browser they inhabit at runtime. So, as the RIA imposes new exchanges with the server, new connections will be opened between the client and the application. This may happen because of prefetching, polling, or state synchronization, among others. Because of these, the servers may see an increase in TCP load (including number of concurrent TCP connections and setup/teardown rate) if applications migrate from the traditional model to an RIA.

- **More HTTP Transactions -** The compartmentalization of applications due to the nature of RIAs allows them to fetch portions of a web page from the server in pieces – this is true whether items are requested in real time or prefetched. State synchronization will also require extra transactions between the client and the server. Likewise, polling mechanisms will require an HTTP exchange with every iteration. For example, an application like Google Suggest will require a transaction with every keystroke. So, even if the size of pages may decrease, the number of exchanges will likely increase. The application will now see many small HTTP transactions, rather than a few big ones. This may cause a new strain on the servers and any proxy elements between the browser and the application.

- **Larger First Pages -** Obviously, the browser in an RIA doesn't become "smarter" on its own. The newly added intelligence on the client side is enabled by extra code, whether it's by Javascript or Flash. This extra code will either be embedded in the page itself, or fetched by the page as an embedded object. Either way, the start point of the application will be significantly larger in an RIA than in a traditional web application.

- **Higher Bandwidth Requirements -** RIAs will vary from deployment to deployment. But, they have the potential to be "chatty" if the amount of communication between the client and the server is continuous and vital to the operation of the application. Anticipating through prefetching, for example, requires many more object retrievals from the server; objects the user may never need or access. In such cases, the infrastructure may need to provide a higher level of bandwidth for the application – something that will affect the network and the servers. Although the impact of these issues will vary from application to application, it's important to understand that adding intelligence to the client side of a web application may have some incurred costs. This will help better prepare deployment and use of these new, powerful applications.

RIAs and Crescendo's AppBeat DC

Crescendo's AppBeat DC is an Application Delivery Controller (ADC) that front-ends web applications and provides powerful acceleration, optimization, and availability features for the application. AppBeat DC is built on a purely hardware-based platform

deploying purpose-built hardware components to offer various services to the application. This architectural model provides significant scale for the platform as a whole, allowing it to optimize traffic at multi-gigabit speeds while handing TCP connections and HTTP requests at massive rates. Furthermore, the platform's distributed architecture provides superior feature concurrency, allowing a user to enable multiple features at once without any performance degradation. AppBeat DC provides services such as TCP offload and multiplexing, load balancing, content compression, and SSL offload for web applications. These services can also be used to help enhance RIAs, since they are also built on a transactional model and use standard HTTP for client-to-server communication. AppBeat DC's powerful hardware platform makes it an optimal choice for accelerating and optimizing RIAs, especially when it comes to migrating from traditional applications to RIAs. AppBeat DC's enormous capacity allows it to handle large amounts of traffic and high transaction rates with ease. AppBeat DC can provide a number of significant benefits for RIAs:

- **High Transaction Rate -** When migrating from a traditional application to an RIA, the server infrastructure may experience a sharp increase in transaction rates for reasons discussed above. AppBeat DC's offload functionality significantly scales the capacity of a server, allowing it to handle many more transactions per second. So, the demands of a newly deployed RIA can be easily met by deploying AppBeat DC in front of the server infrastructure.

- **Connection Shielding -** As discussed above, the TCP load on a server infrastructure will likely increase after migrating to RIAs since the client-side needs to communicate more frequently with the application. AppBeat DC is the TCP termination point for all incoming client connections and its hardware platform allows it to handle these connections optimally and in significant quantities. AppBeat DC owns and controls TCP connections to the servers, multiplexing the many client-side connections to the few long lasting server-side connections. As a result, the servers are never exposed to the higher connection volumes and continue to operate as before. AppBeat DC takes over the handling and processing of all new connections.

- **Compression** - Much of the communication between the client-side and server-side components of an RIA are text based. Javascript code and XML communication (if used) are both highly compressible. AppBeat DC's hardware- based compression engine can compress all relevant content as they flow from the servers to the clients. This is done at gigabit speeds and with zero latency. This helps the overall response time of the RIA (when the transactions are real time). At the same time, since compression reduces the overall outbound bandwidth of the application, AppBeat DC can help alleviate some of the new bandwidth demands that the RIA may place on the network.

- **Load Balancing -** As with any application, the server side of an RIA will have availability and scale requirements that may necessitate deploying multiple servers. In these cases, AppBeat DC's advanced load balancing

functionality can provide the proper fault tolerance, while ensuring that client requests are always sent to the most available and appropriate server.

- **SSL Offload -** RIAs may, and will, still use secure connections from the browser to the server for exchanging sensitive data. Since the client-side of an RIA uses the browser's inherent functionality, this will be done via SSL. As such, the SSL processing burdens will still be an issue for servers, even in an RIA deployment. AppBeat DC's SSL offload functionality can relieve the servers from having to deal with these high-impact SSL sessions. SSL sessions are terminated at AppBeat DC and processed for server delivery. Secure content can still benefit from all AppBeat DC services such as load balancing and compression.

Conclusion

Rich Internet Applications are taking the web into exciting new directions. As the traditional web application model shifts to that of an RIA, we see more powerful user interfaces, and a richer client experience due to the "smartening" of the browser side of an application. As exciting as RIAs are, they introduce some new concepts and issues that need to be taken into account when examining them. These elements are not only important to the application deployment itself, but also to the network and server infrastructure that hosts them. Traffic profiles change when RIAs are deployed, either slightly or significantly, depending on the RIA. The same is true of the way in which clients and servers communicate. These issues are all important to network administrators as they evaluate the impact of an RIA on their network and servers.

Crescendo's AppBeat DC accelerates and optimizes web applications. With its massive capacity, AppBeat DC front-ends applications of varying size, from environments that require tens of Mbps to those that require multi-gigabit deployments. However, AppBeat DC is also an optimal solution for accelerating and optimizing RIAs. The solution can handle with ease any extra load that may be placed on the network, while shielding the servers and allowing them to operate in an optimized environment. At the same time, AppBeat DC's optimization services can help the RIA itself and allow it to server more clients, with better response times.

About Crescendo Networks

Crescendo Networks is the recognized performance leader for accelerating and optimizing the delivery of business-critical, Web-enabled applications. The company's unique multi-tier application delivery architecture dramatically improves the operation of today's demanding application infrastructure. The world's largest corporations and fastest growing Web properties rely on Crescendo for the application performance and efficiency needed to ensure usability, facilitate rapid business growth, lower IT costs and capture additional revenue. To learn more about Crescendo Networks' application delivery solutions, visit www.crescendonetworks.com.