HTML5 Web Sockets Changes the Scalability Game

Lori MacVittie, 2011-07-11

#HTML5 *Web Sockets are poised to completely change scalability models … again.*

Using Web Sockets instead of XMLHTTPRequest and AJAX polling methods will dramatically reduce the number of connections required by servers and thus has a positive impact on performance. But that reliance on a single connection also changes the scalability game, at least in terms of architecture.

Here comes the (computer) science…

If you aren't familiar with what is sure to be a disruptive web technology you should be.

Web Sockets, while not broadly in use (it is only a specification, and a non-stable one at that) today is getting a lot of attention based on its core precepts and model.

Web Sockets

> Defined in the Communications section of the HTML5 specification, HTML5 Web Sockets represents the next evolution of web communications—a full-duplex, bidirectional communications channel that operates through a single socket over the Web. HTML5 Web Sockets provides a true standard that you can use to build scalable, real-time web applications. In addition, since it provides a socket that is native to the browser, it eliminates many of the problems Comet solutions are prone to. Web Sockets removes the overhead and dramatically reduces complexity.
>
> - HTML5 Web Sockets: A Quantum Leap in Scalability for the Web

So far, so good. The premise upon which the improvements in scalability coming from Web Sockets are based is the elimination of HTTP headers (reduces bandwidth dramatically) and session management overhead that can be incurred by the closing and opening of TCP connections. There's only one connection required between the client and server over which much smaller data segments can be sent without necessarily requiring a request and a response pair.

That communication pattern is definitely more scalable from a performance perspective, and also has a positive impact of reducing the number of connections per client required on the server. Similar techniques have long been used in application delivery (TCP multiplexing) to achieve the same results – a more scalable application. So far, so good.

Where the scalability model ends up having a significant impact on infrastructure and architectures is the longevity of that single connection:

> Unlike regular HTTP traffic, which uses a request/response protocol, WebSocket connections can remain open for a long time.
>
> - How HTML5 Web Sockets Interact With Proxy Servers

This single, persistent connection combined with a lot of, shall we say, interesting commentary on the interaction with intermediate proxies such as load balancers. But ignoring that for the nonce, let's focus on the "remain open for a long time."

A given application instance has a limit on the number of concurrent connections it can theoretically and operationally manage before it reaches the threshold at which performance begins to dramatically degrade. That's the price paid for TCP session management in general by every device and server that manages TCP-based connections.

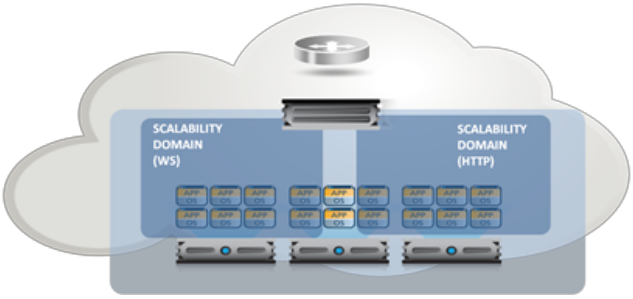
But Lori, you're thinking, HTTP 1.1 connections are persistent, too. In fact, you don't even have to tell an HTTP 1.1 server to keep-alive the connection! This really isn't a big change.

Whoa there hoss, yes it is. While you'd be right in that HTTP connections are also persistent, they generally have very short connection timeout settings. For example, the default connection timeout for Apache 2.0 is 15 seconds and for Apache 2.2 a mere 5 seconds. A well-tuned web server, in fact, will have thresholds that closely match the interaction patterns of the application it is hosting. This is because it's a recognized truism that long and often idle connections tie up server processes or threads that negatively impact overall capacity and performance. Thus the introduction of connections that remain open for a long time changes the capacity of the server and introduces potential performance issues when that same server is also tasked with managing other short-lived, connection-oriented requests.

Why this Changes the Game…

One of the most common inhibitors of scale and high-performance for web applications today is the deployment of both near-real-time communication functions (AJAX) and traditional web content functions on the same server.

That's because web servers do not support a per-application HTTP profile. That is to say, the configuration for a web server is global; every communication exchange uses the same configuration values such as connection timeouts. That means configuring the web server for exchanges that would benefit from a longer time out end up with a lot of hanging connections doing absolutely nothing because they were used to grab standard dynamic or static content and then ignored. Conversely, configuring for quick bursts of requests necessarily sets timeout values too low for near or real-time exchanges and can cause performance issues as a client continually opens and re-opens connections. Remember, an idle connection is a drain on resources that directly impacts the performance and capacity of applications. So it's a Very Bad Thing™.



One of the solutions to this somewhat frustrating conundrum, made more feasible by the advent of cloud computing and virtualization, is to deploy specialized servers in a scalability domain-based architecture using infrastructure scalability patterns. Another approach to ensuring scalability is to offload responsibility for performance and connection management to an appropriately capable intermediary.

Now, one would hope that a web server implementing support for both HTTP and Web Sockets would support separately configurable values for communication settings on at least the protocol level. Today there are very few web servers that support both HTTP and Web Sockets. It's a nascent and still evolving standard so many of the servers are "pure" Web Sockets servers, many implemented in familiar scripting languages like PHP and Python. Which means two separate sets of servers that must be managed and scaled. Which should sound a lot like … specialized servers in a scalability domain-based architecture.

The more things change, the more they stay the same.

The second impact on scalability architectures centers on the premise that Web Sockets keep one connection open over which message bits can be exchanged. This ties up resources, but it also requires that clients maintain a connection ***to a specific server instance.***

This means infrastructure (like load balancers and web/application servers) will need to support persistence (not the same as persistent, you can read about the difference here if you're so inclined). That's because once connected to a Web Socket service the performance benefits are only realized if you ***stay*** connected to that same service. If you don't and end up opening a second (or Heaven-forbid a third or more) connection, the ***first*** connection may remain open until it times out. Given that the premise of the Web Socket is to stay open – even through potentially longer idle intervals – it may remain open, with no client, until the configured time out. That means completely useless resources tied up by … nothing. Persistence-based load balancing is a common feature of next-generation load balancers (application delivery controllers) and even most cloud-based load balancing services. It is also commonly implemented in application server clustering offerings, where you'll find it called ***server-affinity***. It is worth noting that persistence-based load balancing is not without its own set of gotchas when it comes to performance and capacity.

THE ANSWER: ARCHITECTURE

The reason that these two ramifications of Web Sockets impacts the scalability game is it requires an broader architectural approach to scalability. It can't necessarily be achieved simply by duplicating services and distributing the load across them. Persistence requires collaboration with the load distribution mechanism and there are protocol-based security constraints with respect to incorporating even intra-domain content in a single page/application. While these security constraints are addressable through configuration, the same caveats with regards to the lack of granularity in configuration at the infrastructure (web/application server) layer must be made. Careful consideration of what may be accidentally allowed and/or disallowed is necessary to prevent unintended consequences. And that's not even starting to consider the potential use of Web Sockets as an attack vector, particularly in the realm of DDoS. The long-lived nature of a Web Socket connection is bound to be exploited at some point in the future, which will engender another round of evaluating how to best address application-layer DDoS attacks.

A service-focused, distributed (and collaborative) approach to scalability is likely to garner the highest levels of success when employing Web Socket-based functionality within a broader web application, as opposed to the popular cookie-cutter cloning approach made exceedingly easy by virtualization.

- Infrastructure Scalability Pattern: Partition by Function or Type
- Infrastructure Scalability Pattern: Sharding Sessions
- Amazon Makes the Cloud Sticky
- Load Balancing Fu: Beware the Algorithm and Sticky Sessions
- Et Tu, Browser?
- Forget Hyper-Scale. Think Hyper-Local Scale.
- Infrastructure Scalability Pattern: Sharding Streams
- Infrastructure Architecture: Whitelisting with JSON and API Keys
- Does This Application Make My Browser Look Fat?
- HTTP Now Serving … Everything

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc. Corporate Headquarters info@f5.com	F5 Networks Asia-Pacific apacinfo@f5.com	F5 Networks Ltd. Europe/Middle-East/Africa emeainfo@f5.com	F5 Networks Japan K.K. f5j-info@f5.com