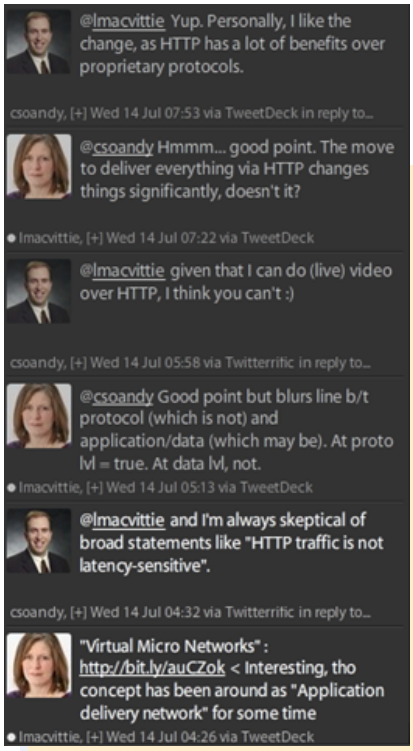
HTTP Now Serving … Everything

Lori MacVittie, 2010-20-07

You can't assume anything about an application's performance and delivery needs based on the fact that it rides on HTTP.



I read an interesting article during my daily perusal of most of the Internet (I've had to cut back because the Internet is growing faster than my ability to consume) on "Virtual Micro Networks."

> The VMN concept goes well beyond Virtual Local Area Networks (VLANs). Like VLANs or any other network, VMNs transport data from source to destination. But VMNs extend beyond transport to consider security, location, users, and applications. VMNs address:
>
> Where is the information? The answer to this question used to be a physical server or storage device but application switching and server/storage virtualization makes this more dynamic and complex.
>
> […]
>
> VMNs also must be aware of traffic type. For example, voice, video, and storage traffic is extremely latency-sensitive while HTTP traffic is not. Additionally, some network traffic may contain confidential information that should be encrypted or even blocked.
>
> What are the specific characteristics of the information? Network-based applications may be made up of numerous services that come together at the user browser. How they get there isn't always straightforward, thus the rise of vendors like Citrix NetScaler and F5 Networks. This is also where security comes into play as certain traffic may be especially sensitive, suspicious, or susceptible. [emphasis added]
>
> What's driving creation of Virtual Micro Networks

Okay, so first things first: the author really is using another term to describe what we've been calling for some time an application delivery network. That was cool in and of itself; not the emergence of yet another TLA but that the concept is apparently out there and rising. But what was even more interesting was the conversation this started on Twitter. If you don't follow @csoandy (that's the Twitternym of Andy Ellis of Akamai Networks) you might want to start. Andy pointed out that the statement "HTTP traffic is not latency-sensitive" is a bit too broad and went on to point out that it really depends on what you're delivering. Live video, after all, ***is*** sensitive to latency no matter what protocol is transporting it.

Andy put it well in an off-line conversation when he said, "There's also the myth that HTTP isn't for low latency apps. HTTP lets you take advantage of optimizations done in TCP and HTTP to accelerate delivery." A great point and very true. All the "built-in acceleration and optimization" of an application delivery controller's TCP stack is free for HTTP because after all, HTTP rides on TCP. But ironically this is also where things get a bit wonky.

The reality is that the application ***data*** is sensitive, not the protocol. But because the data HTTP was initially designed to transport was ***not*** considered to be latency sensitive, you almost have to look at HTTP as though ***it is,*** which is why the broad statement bothered Andy in the first place. We wouldn't say something like "TCP" or "UDP" is not sensitive to latency because these are transport layer protocols. We need to know about the data that's being transported. Similarly, we can't (anymore) say "HTTP isn't sensitive to latency" because HTTP is the de facto transport protocol of web applications. As I remarked to Andy, the move to deliver everything via HTTP changes things significantly. "Things" being the entire realm of optimization, acceleration, and application delivery.

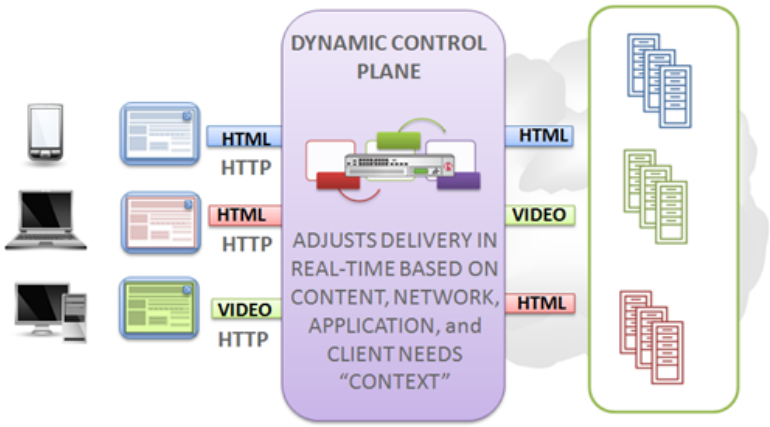
Context is Everything

As the initial blog post that started this conversation pointed out, and which nothing Andy and I discussed really changed, is that our nearly complete reliance on HTTP as the de facto transport protocol for everything means that the infrastructure really needs to be aware of the context in which requests and responses are handled.

When an HTTP GET and its associated response might in one case be a simple binary image and in another case it might be the initiation of a live video stream, well, the infrastructure better be able to not only recognize the difference but *handle them differently.* HTTP doesn't change regardless, but the delivery needs of the data do change.

This is the "application aware" mantra we (as in the entire application delivery industry) have been chanting for years. And now it's becoming an imperative because HTTP no longer implies text and images, it implies *nothing*. The infrastructure responsible for delivering (securing, optimizing, accelerating, load balancing) the access to that application data cannot assume anything; not session length, not content length, not content type, not optimal network conditions. The policies that might ensure a secure, fast, and available web application are almost certainly not the same policies that will provide the same assurance for video, or audio, or even lengthy binary data. The policies that govern the delivery of a user-focused application are not the same ones that should govern the delivery of integration-driven applications like Web 2.0 and cloud computing APIs. These are different data types, different use cases, different needs.

Yet they are all delivered via the same protocol: HTTP.



Dynamic Infrastructure Needed

What makes this all even more complicated (yes, it does get worse as a matter of fact) is that not only is the same protocol used to deliver different types of data but in many cases it may be delivered to the same user in the same session.

A user might move from an article or blog to a video back to text all the while a stream of Twitter and Facebook updates are updating a gadget in that application. And the same infrastructure has to handle all of it. Simultaneously.

Wheeeeeeeeeee!

That HTTP can be extended (and has been, and will continue to be) to include broad advanced capabilities has been a blessing and a curse, for as we deliver more and more differenter content over the same protocol the infrastructure must be able to ascertain dynamically what type of data is being delivered and apply the appropriate actions dynamically. And it has to incorporate user information, as well, because applications highly sensitive to latency need special care and feeding when delivered over a congested, bandwidth constrained network as opposed to delivery via a high-speed, low latency LAN. The application delivery network, from user to application and back, must be context-aware and able to "turn on a dime" as it were, and adjust delivery policies based on conditions at the time of the request and subsequent responses. It's got to by dynamic.

Consider the comparison offered by Andy regarding video served via traditional protocols and HTTP:

> Consider a live stream; say, Hope for Haiti. A user opens a browser, and has a small embedded video, with a button to expand to full screen. With most streaming protocols, to get a higher resolution stream, your player needs to either:
>
> a) start grabbing a second, high res stream in the background, and guess when to splice them over. (now consider if the stream is too fat, and you need to downgrade)
>
> b) pause (drop existing stream) and grab a new stream, exposing buffering to a user.

> c) signal somehow to the streaming server that it should splice in new content (we built this. it's *hard* to get right. And you have to do it differently for each protocol).
>
> With HTTP, instead what you see is:
>
> a) Browser player is grabbing short (usually 2) second chunks of live streaming content. When it detects that it is fuller screen, and, inferring available bandwidth by how long it takes to download a chunk, ask for a higher resolution chunk for the next available piece.

Quite the difference, isn't it? But underlying that simplicity is the ubiquity of HTTP and a highly dynamic, flexible infrastructure capable of adapting to the sensitivities specific not only to the protocol, to the data, but to the ***type*** of data being delivered.

So it turns that both Andy and I are both right, it just depends on how you're looking at it. It isn't that HTTP is sensitive to latency, it isn't. But the data being delivered over HTTP most certainly is. But it ***is*** confusing to discuss HTTP in broad, general terms because you can't assume anymore that what's being delivered is text and images. We don't talk in terms of TCP when we talk web applications, so maybe it's time to stop generalizing about "HTTP" and start focusing on applications and data, about the ***content,*** because that's where the real challenges surrounding performance and security are hiding.

Related Posts

- What's driving creation of Virtual Micro Networks
- HTTP: The de facto application transport protocol of the Web

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