iRules and Wait States, after Makes Life Better

Colin Walker, 2010-09-11

In my years working on the DevCentral team I've gotten all sorts of crazy requests for iRules. Whether it's an engineer wanting to perform a specific kind of persistence via an array in an iRule for roughly 3 million concurrent connections, or someone wanting to send the Homer Simpson "Doh!" sound file via an iRule as part of their error page, you'd be surprised at the kind of things I see come through. One of my favorite stories is about trying to inject delay into a connection. Why is this one of my favorite stories? Because it's a lot more brutal than it sounds, simple as it is.

The iRule itself is extremely simple, and I'll re-post it here with a *very strong caveat*: **DO NOT USE THIS iRULE**. How's that for strong? The intent of this iRule was to make the LTM into a latency insertion engine. This isn't the actual iRule from years ago, but it's pretty similar. Basically the engineer wanted to insert a certain amount of latency into every connection to simulate a WAN link. The iRule worked completely as intended.

```
1: when HTTP_REQUEST {
2:   set newtime [expr [clock -miliseconds] + 50]
3:   while {[clock -milliseconds] >= $newtime } {
4:   }
5: }
```

"So why the problem?" you ask? Because it also completely fouled up all the other testing on the system that said engineer didn't know about. See, because of the single-threaded nature of TMM, an iRule like the above is very painful. It forces each and every request going through that iRule to send TMM into a holding pattern until it completes, unable to process any other traffic in the mean time. This may not be a huge deal if you're dealing with only your own, finite set of traffic, but what if you're on a shared testing resource and there are 20 other people trying to run tests against other virtuals? Even though their traffic isn't going through your iRule, it's still feeling the pain of the delay, and it can foul things up rather badly. At the very least it's going to have them pulling their hair out, trying to figure out why their response times just jumped up out of nowhere.

So what to do then, if you want your system to be able to perform this, or any other task that requires a wait loop? Well, as is often the case, the genius minds on the F5 PD team have already solved this problem for us, and I didn't really think about it until an email came across an internal mailing list mentioning this use of the after command recently. The after command sends things into a wait state as well, but it does so on a *per connection* basis, rather than locking out the entire TMM. What you end up with is the connection being delayed as desired, to insert the delay, but the TMM is freed up immediately after processing the after command to continue dealing with traffic as normal. So instead of a box that's completely bound up in while loops, your box is processing traffic free and easy.

```
1: when HTTP_REQUEST {
2:    after 50
3: }
```

Does this mean LTM is now your go-to WAN simulation device? Well..no, but it means that you can now perform this, and any other various tasks that might insert some kind of delay into a connection, without fear of locking up all your TMM cores and messing with the traffic being processed by everyone else. For more on the awesome after command, be sure to check out the Wiki as there are a few different iterations of the command and some good examples to be found there.

Related Articles

- v.10 - iRules and the after command > DevCentral > F5 DevCentral ...
- DevCentral Wiki: after
- Colin Walker - after

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