iRules Concepts: Tcl, The How and Why

Colin Walker, 2012-13-02

F5 uses Tcl as the interpreter for iRules. Many people often ask why that is. This questions is usually followed up by an immediate, "Why not Perl?" or "Why not Java" or "Why not <fill in my preferred language of choice>?". I understand the question, and frankly I'm a Perl guy from way back myself, so when I first landed at F5 and started devouring all things iRules, I was curious about the same thing. Since then I've discussed this topic with some of F5's best, in my opinion, and have come to understand that there are many solid reasons for choosing the runtime that we use.

When asked "Why Tcl?" my standard response centers around varying degrees of discussing:

- Speed
- Embeddability
- Usability

These all remain true today, and I will expand on each of them in hopes of illuminating our position with Tcl and iRules, and why the Perl lover in me was, and is to this day, convinced that we made the right choice.

History

Before I delve into the above list, first let me give some history of how we originally got to Tcl in the first place. Originally, back in the days of dinosaurs and BIG-IP v3.0, when iRules was introduced as a technology within F5, we used a custom syntax. We hand rolled commands and utility functions, and relied on no particular language to achieve this, other than C, which is the base for pretty much everything that runs on our box, and frankly darn near everywhere else. This was all well and good, but as any F5 historian knows, the world changed a lot for us in v9.

For version 9 we tore up pretty much everything on the drawing board. Heck, we tore up the drawing board and started fresh. Even still, the plan was to go forward with the same approach, I.E. custom roll commands to be used within iRules and rely on no outside language to achieve this. At this point, though, some testing was done and a shocking result surfaced. In testing Tcl against the custom built commands, Tcl was actually faster in many cases. (More about how and why in a bit.) This left us with a very interesting fact: Tcl was both faster and more feature rich than our hand rolled commands. That made the choice pretty simple.

Now then, on to the reasons why Tcl was and still is a solid choice for iRules. From the list above, let’s start with speed.

Speed

We talk about it all the time. iRules are fast. Think of the fastest thing you can think of, a speeding bullet perhaps. Now think faster. No, faster than that. Seriously, blazingly fast. Why so much emphasis on speed? Because it lies at the core of everything that we do.

I’m one of the first people to say that it’s not all about speeds and feeds, as it were, when talking about ADC solutions from a broad perspective. That is not to be confused with the concept that speed is not important. Speed, in fact, is absolutely paramount when speaking from a granular level. The faster the granular functions are; packet interrogation, re-writing, routing, forwarding, IP translation, the faster each atomic function is, the less resources are utilized on such things. This leaves more power available as overhead to build and complete complex logical functions. Building business logic into the network could be costly to a point of making it prohibitive if it weren’t for the fact that each minute operation being performed behind the scenes were streamlined to the nth degree. So when it comes to iRules or any programmable interface for the network, speed is absolutely paramount.

That’s all well and good, but how does Tcl fit into that? In testing Tcl against other, heavier languages such as Perl and Python, we deemed pretty quickly two things:

1) Tcl was far, far faster for our purposes than any of the other widely available options at the time, which is still the case with the exception of perhaps Lua.

2) Other options had large amounts of commands that we would either not need, or explicitly did not want to include for either security or performance reasons.

So out of the box Tcl is a faster choice for our needs. If that weren't enough, however, we also have to take into account the fact that we need to heavily modify the functionality of the language. For our purposes we both add and rip out large chunks of commands and functionality. We need to make things network aware, event driven (which, by the way, Tcl explicitly is, and most other languages are not), add the notion of suspending/parking commands, ensure that garbage collection doesn't occur in the middle of processing network traffic, and generally twist the language into something that understands what we are doing. Tcl is very easy to modify in all of these ways, compared to other options, so this one more way in which it is a good fit for our needs.

Also keep in mind that the reality is, we're only using a very small slice of what is available in whatever language iRules makes use of. This is because the vast majority of iRules commands are actually custom functions being performed within the TMM. Things like the HTTP:: commands, the table command, class, iStats, sideband connections … by far the lion's share of what iRules users rely on for the functionality to understand, interpret and modify their traffic are actually calls to native C code. That is, those things don't actually exist at all in the Tcl world, beyond creating a Tcl wrapper to call and handle the underlying functions.

This is for various reasons, not the least of which is – you guessed it – performance. The functions within TMM that perform these actions are far higher performance in their native state than they could be in any interpreted language running per connection, Tcl or otherwise. So if a huge majority of our commands are actually just passing control back and forth between Tcl and C, then a highly adept interface for doing so becomes paramount. Tcl, again, pulls ahead of the pack in this arena. It just so happens that Tcl has one of the more thorough C programming APIs available, compared to other similar language options. Given how often this happens and how important it is to what iRules does at its core, this is a big plus.

Last but not least, Tcl supports the notion of compiling to byte-code. This is something we make extensive use of to boost performance at run time. Whenever an iRule is saved to the system it is compiled into byte code, which allows it to execute far faster than if it were in the native, human readable state. Most scripting languages combine the compilation and execution functions so that both occur effectively at the same time. With Tcl we're able to use a different model that allows for the compilation, syntax checking etc. to occur at load time, which means that at run time, the byte code is processed instead of the original iRule, thus skipping a large amount of the overhead that would otherwise be involved. This allows a far smaller footprint at run time (meaning when the iRule has to execute), in exchange for a bit of extra work at load time (when a user saves an iRule), which is a very, very solid trade for us. Anything we can offload to happen once at load time rather than for each connection that comes through the system is an extremely solid performance improvement. 1 execution per save vs 100k executions per second (on a highly traffic laden box) is a pretty simple picture to understand, and bytecode allows us to achieve that at least somewhat.

Embeddability

Tcl is not only extremely fast, but also supremely embeddable. It has a long history of being a go-to embedded interpreter in many fast paced, low level systems such as L2 switches. This is thanks to the fact that Tcl is very, very small, when compared to other languages that offer similar functionality (or more functionality, like Perl and Java, but more on that later). Also, Tcl is amazingly simple to integrate with C. So much so that it is considered near free in many cases, and anything written in C could easily be exposed via Tcl with minimal effort. Keep in mind here that when I talk about things written in C, that list includes a massive array of programs and systems, including many modern kernels, such as Windows and Linux. Tcl being friendly with kernels isn't a bad thing when looking at integrating it with a custom micro-kernel, as is the case within TMM.

On top of the highly embeddable nature of Tcl, you also have to factor in the absolutely minuscule footprint. The entirety of Tcl is a few hundred kilobytes, including the parts we're not using within iRules. That is tiny in comparison to its more feature rich cousins Perl and Java and <many others>. For instance the entire source download of Tcl (as of the writing of this article) is 4.3M whereas Perl is 15M. The size of the environment becomes very important the more you understand the inner workings of the iRules world.

One of the things that most people don't take into account or don't realize is the fact that each connection to the BIG-IP that invokes an iRule receives a unique Tcl context along with the accompanying state, variables, etc. This means that memory is allocated to every connection that uses an iRule to store that Tcl structure, allow it to interface uniquely with TMM, and do what it needs for that particular connection and the iRules associated with it. Keep in mind that this can occur millions of times concurrently on a busy, high-end F5 device, and to me it becomes extremely impressive. The memory footprint difference between a couple hundred kilobytes for Tcl and a couple megabytes for many other languages is large enough for a single instance. When you talk about a few hundred thousand or even a million concurrent instances, however, it becomes exponentially larger and more important, as you might imagine.

Surely you could not allocate, store, and process millions of copies of Perl in the same resource footprint. This is directly due to the size and simplicity of Tcl. Perl and other such languages have many, many more base capabilities than Tcl. This is a fantastic thing when and if you need them, and when you aren't worried about resource constraints in such a blisteringly fast paced environment. In our world, when the vast majority of that added functionality isn't needed anyway, and every byte or cycle counts, the overhead isn't nearly worth the luxury.

One of the best engineers here at F5, from whom I gleaned some fantastic insight and new information when asking questions to help inspire this conversation, said it quite well:

"The full Tcl syntax can be described by just a handful of rules. In fact it's so simple you could write your own Tcl grammar parser in an afternoon. For contrast, only Perl can parse Perl."

Don't get me wrong, I'm a big Perl fan, and still use it to this day for writing utilities and such on the command line. There is a tool for every job and in our particular case, given the performance requirements we have, Tcl just makes more sense.

Usability

Now that we understand the performance concerns and how Tcl plays well in that arena, the next most prominent concern on our list would be usability. When considering usability in this case it is important to remember our target audience. The people generally managing these systems are not full time programmers. As such, making use of a simple, easily readable language that is quick to pick up and master, and easy to read and pass from one user to the next makes a lot of sense.

The simplicity of Tcl plays into less overhead to the user when it comes to understanding the commands and tools available just as much as it plays into the system overhead required to load. It makes sense, I think, that a language with far greater capabilities and extended commands, memory structures, modules, etc. would take more time and effort to master. Given that doing so is often not the primary role of the individuals we hope to appeal to with iRules, the simpler approach makes more sense.

On top of that, most of that added functionality simply isn't necessary. Keep in mind we're not even exposing all of Tcl. We intentionally and specifically limit some of the base functionality. If we are limiting the much simpler, less complex language because we don't need or want that functionality, there is little argument for moving to a more complex, feature rich language, given that the majority of the functionality would likely be "nice to have" at best, and undue overhead at worst.

Is Tcl still the right choice?

All of that being said, if we were starting from scratch today without thousands of iRules in the wild, a community built up to support the language, TMSH, iApps and many other technologies within the BIG-IP making use of Tcl, etc. would we choose Tcl again?

Given the needs iRules has for an interpreter: Fast, tightly embeddable, small footprint, fast, easily tied to native C and thus kernels, usability – yes, I think Tcl would still be a strong contender for being the best fit for our needs. If I weren't convinced of it before, I've spoken to some of the core architects responsible for iRules today, and they share the same opinion, that Tcl is still as strong a choice today as it was in 2004. There are others that offer similar benefits today, Lua being chief among them, but there are drawbacks of those as well. To me, there is no clearly superior choice for our very specific needs even in today's landscape. On top of that, we rarely get any complaints about iRules being Tcl based. Sure we get questions as to why, but once we explain the benefits and people are clear on the reasons we went down the road we did, it almost always results in a happy iRules user.

Are there some ways in which allowing users access to other languages may be beneficial? Certainly, but keep in mind they are largely available. It is quite commonplace for Perl and bash to be used for monitors already. Perhaps a better way of addressing the question is: What is it you would like to do with other languages that you cannot via iRules currently? Once that is understood, the discussion could turn to whether or not that is possible, feasible and reasonable to implement within BIG-IP in some fashion. Would this be built directly into the Tcl construct iRules is based on, or in some other fashion that may allow the use of a chosen language, or a subset thereof? Who knows, but it is a valuable conversation regardless of the outcome. The more we can understand what it is people would like to and are trying to do, the better we can continue expanding the already powerful tools that we offer to meet those needs.

Hopefully that paints the picture of why we chose and continue to use Tcl to support our powerful iRules framework. I am not by any means a Tcl zealot. Frankly I had far more experience in other languages before coming to F5, and rather enjoyed writing things in those languages. What I care about far more than writing in my favorite language, however, is using the right tool for the job. As I've learned more about iRules I have come to understand the reasons we use Tcl to do what we do, and appreciate what it allows us.

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

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