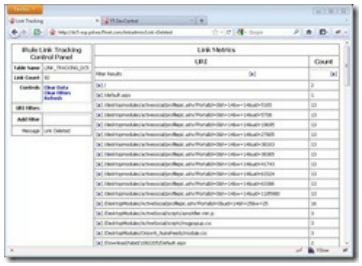
Link Tracking With iRules - Part 3 - URI Filtering

Joe Pruitt, 2011-06-04



This is the third article in a series on building link tracking functionality with iRules. The previous articles were

- Link Tracking With iRules - Part 1
- Link Tracking With iRules - Part 2 - Filtering Results

In the first article, I focused on building the core link tracking logic and in the second I added the logic to implement a simple output filter to only list the URI's you are interested in seeing the statistics for. In this article I will build on additional functionality, so I would highly suggest taking a peek at the first two before continuing.

Now, if you are like me, there are a lot of HTTP requests that go into your application that you don't care about keeping tracking data on. The first two articles did nothing with regards to pre-filtering the URIs coming in to the iRule and left unattended, will get a fairly large list. This article will discuss how to add some basic filtering to the incoming URIs that are added to the link tracking database.

The Control Panel UI Elements

In order to list and store the URI filters in this application, I created a control panel to the left of the URI tracking results.

The control panel consists of the subtable name, the number of links currently in the tracking database, global control links for clearing the data and filters, the URI filter list, and a text box to allow for the user to add new URI filters.

I'd like to note here, that I've expended a *minimal* amount of UI design in this iRule. I have no sense of color, so you are just going to get plain HTML tables out of me. I'd love to get someone with some artistic abilities to fix up the styles in this and make it look nice. Anyone?

Anyway, back to the contents of the control panel..

Table Names

The table names, as in the previous examples, are derived from the name of the virtual server the iRule is attached to. This allows for applications on different virtual servers to have separate data stores. In this example, I'm just storing these as local variables in the HTTP_REQUEST handler event.

```
1: set TABLE_LINK    "LINK_TRACKING_[virtual name]";
2: set TABLE_FILTERS "LINK_TRACKING_FILTERS_[virtual name]";
```

Link Count

The number of items in the link tracking database is easily determined by using the "**table keys**" command with the "**-count**" argument. This will return the number of keys that would have been returned from the command if the "**-count**" parameter wasn't specified.

```
1: set count [table keys -subtable $TABLE_LINK -count]
2: append content "<tr><td align='right'><b>Link Count</b></td><td>$count</td></tr>"
```

Controls

I've moved the "Clear Data" control from the report listing into the Controls section of the control panel. What that I know have three control commands:

- /linkcleardata - Clear all link tracking data from the table. In the handler for this command, I simply issue the "**table delete**" command on the *$TABLE_LINK* table.
- /linkclearfilters - Clear all filters from the filter table. In this handler, I use the "**table delete**" command to erase the *$TABLE_FILTERS* table.
- /linkadmin - reload the iRule to refresh the result table.

```
1: append content {<tr><td align='right' valign='top'><b>Controls</b></td>
```

```
  <td>
    <a href='/linkcleardata'>Clear Data</a><br/>
    <a href='/linkclearfilters'>Clear Filters</a><br/>
    <a href='/linkadmin'>Refresh</a>
  </td>
</tr>}
.
.
"/linkcleardata" {
  table delete -subtable $TABLE_LINK -all;
  HTTP::redirect "http://[HTTP::host]/linkadmin/Link+Tracking+Cleared"
}
"/linkclearfilters" {
  table delete -subtable $TABLE_FILTERS -all;
  HTTP::redirect "http://[HTTP::host]/linkadmin/Link+Filters+Cleared"
}
```

Adding New URI Filters

Adding new filters is relatively simple. An HTML **input** form is created in the control panel and I've overridden the **onKeyDown** handler to look for the enter key (keyCode == 13) and when it's hit, the page is redirected to the "/linkaddfilter" command passing in the new filter as the rest of the URI.

The add filter processing code will extract all the values past the initial "/linkaddfilter/" in the URI and insert that into the *$TABLE_FILTERS* table. This table is just being used to store the filter patterns. In this case I really don't care what the value is, nor if it's already in the table so a single "table add" command is made to insert it for future processing.

```
append content {<tr><td align='right'><b>Add Filter</b></td><td>
  <input type='text' id='new_filter' value='' size='15'
    onkeydown="if (event.keyCode == 13) { window.location.assign('/linkaddfilter/' + encodeURI(getElementById('new_filter').value)) }"/>
  </td></tr>}
.
.
"/linkaddfilter/*" {
  set f [string range [HTTP::uri] [string length "/linkaddfilter/"] end]
  if { "" != $f } {
    table add -subtable $TABLE_FILTERS $f 1 indefinite indefinite;
  }
  HTTP::redirect "http://[HTTP::host]/linkadmin/Filter+Added";
}
```

Listing And Removing URI Filters

As I mentioned in the previous section, the URI filters are stored in the *$TABLE_FILTERS* table. The UI listing for these filters simply iterates through all the keys in the table (which are the filters) and inserts them in a list. Before the actual filter string, I've inserted a "[x]" link that will allow for removing the filter in a similar way that I did for removing individual URIs from the link tracking database.

If the delete link is clicked, the filter is extracted from the URI and the "**table delete**" command is called with that filter as the key. A HTTP redirect is then made to refresh the page and show the updated list of URI filters.

```
append content "<tr><td align='right' valign='top'><b>URI Filters</b></td><td>"
foreach key [table keys -subtable $TABLE_FILTERS] {
  append content "\[<a href='/linkremovefilter/$key'>x</a>\] $key<br/>";
}
append content "</td></tr>";
.
.
"/linkremovefilter/*" {
  set val [string range [HTTP::uri] [string length "/linkremovefilter/"] end]
  if { "" != $val } {
    table delete -subtable $TABLE_FILTERS $val;
  }
  HTTP::redirect "http://[HTTP::host]/linkadmin/Filter+Deleted";
}
```

Processing The URI Filters

The default handler for all URIs that are not handled by the iRule previously inserted all URIs with the "**table**" command. I've added additional logic to look for the existence of any filters and process them. I'm using the *$match* variable to determine if the URI is to be stored. If there are no filters in the filter table, then I'm assuming that all URIs will get added to the tracking database. But, if there are some filters in the filter table, I iterate through each of them with the "**foreach**" command and use the "string match" command to see if the incoming URI matches each individual filter. The "**string match**" command is a great tool for performing wildcard comparisons. It is much lighter weight than regular expressions and usually does just what you need. If a match occurs, I break out of the iteration and proceed to entering the value into the tracking database.

```
default {
  set match 1;
  set c [table keys -subtable $TABLE_FILTERS -count]
  if { $c != 0 } {
```

```
    set match 0;
    foreach key [lsort [table keys -subtable $TABLE_FILTERS]] {
      set m [string match $key [HTTP::uri]];
      if { 1 == $m } {
        set match 1;
        break;
      }
    }
  }
  if { $match ==  1} {
    if { [table incr -subtable $TABLE_LINK -mustexist [HTTP::uri]] eq ""} {
      table set -subtable $TABLE_LINK [HTTP::uri] 1 indefinite indefinite;
    }
  }
}
```

Video Walkthrough

Link Tracking With iRules - Part 3 - URI Filtering

Conclusion

So, with the addition of this article, you you not only have the ability to tracking and report on link metrics, but you can only store data on the ones that you are concerned about. The logic for the filter table could easily be modified to use it as a negative filter (ie. allow URI's that **don't** match the filters) but I'll leave that for the reader to develop.

Still More To Come!

I'm not done yet. In the next addition to this iRule, I will add more advanced functionality to allow tracking external referral links.

Get The Source

The source code for this application can be downloaded the LinkTracking3 topic in the iRules CodeShare.

Related Articles on DevCentral

- F5 DevCentral > Hot Topics > iRules
- DevCentral Wiki: iRules Reference
- iRules Event Order > DevCentral > F5 DevCentral > Tech Tips
- Help Kill IE6 with iRules > DevCentral > F5 DevCentral > Tech Tips
- Development Performance Metrics Will Eventually Favor Cost per ...
- Link Tracking With iRules - Part 2 - Filtering Results ...
- Link Tracking With iRules - Part 1 > DevCentral > F5 DevCentral ...
- iRules: Rewriting URIs for Fun and Profit
- Heatmaps, iRules Style: Part 1 > DevCentral > F5 DevCentral > Tech ...
- v10.1 - The table Command - The Basics > DevCentral > F5 ...
- Custom Reporting with iRules > DevCentral > F5 DevCentral > Tech Tips
- iRules 101 #15 - TCL List Handling Commands > DevCentral > F5 ...

Technorati Tags: link, tracking, table, metrics, reporting, irules

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