BIG-IP Interface Stats in Real Time with a TMSH Script

Jason Rahm, 2013-17-09

For the savants among us, calculating bits in and bits out over a delta from two snapshots of the interface counters is a walk in the park. For the rest of us, it's nice to have a tool to look at the current traffic load on an interface while working in the command line interface. This article will walk you through creating a TMSH script to do just that.

Source Data

You can get at interface data via snmp and icontrol, but is also available with the **tmsh show net interface** command.

```
-------------------------------------------------------
Net::Interface
Name  Status   Bits  Bits  Pkts  Pkts  Drops  Errs  Media
                In   Out    In   Out
-------------------------------------------------------
1.1       up  59.4T  5.0T  6.2G  2.4G      0     0   none
```

Yep, that's data. But when you get to terabits, the dial doesn't move quite so quickly, so taking a diff every few seconds won't amount to much. Specifying the **raw** option on the show net interface command helps out in that regard.

```
 (raw)
-------------------------------------------------------------------------------------------
Net::Interface
Name    Status            Bits           Bits        Pkts        Pkts  Drops  Errs  Media
                            In            Out          In         Out
-------------------------------------------------------------------------------------------
1.1         up  59485486972968  5080727699544  6291600606  2488751052      0     0   none
```

That's better, but a little more challenging to parse than adding the **field-fmt** option, which puts it in a nice key value pair list. The bits-in and bits-out counters are the focus of this script.

```
net interface 1.1 {
    counters.bits-in 59486479580896
    counters.bits-out 5080875828888
    counters.drops-all 0
    counters.errors-all 0
    counters.pkts-in 6291722759
    counters.pkts-out 2488812198
    media-active none
    name 1.1
    status up
}
```

Now that we have key value pairs, and already separated by whitespace, this is a simple extraction once we split the entire string by newline.

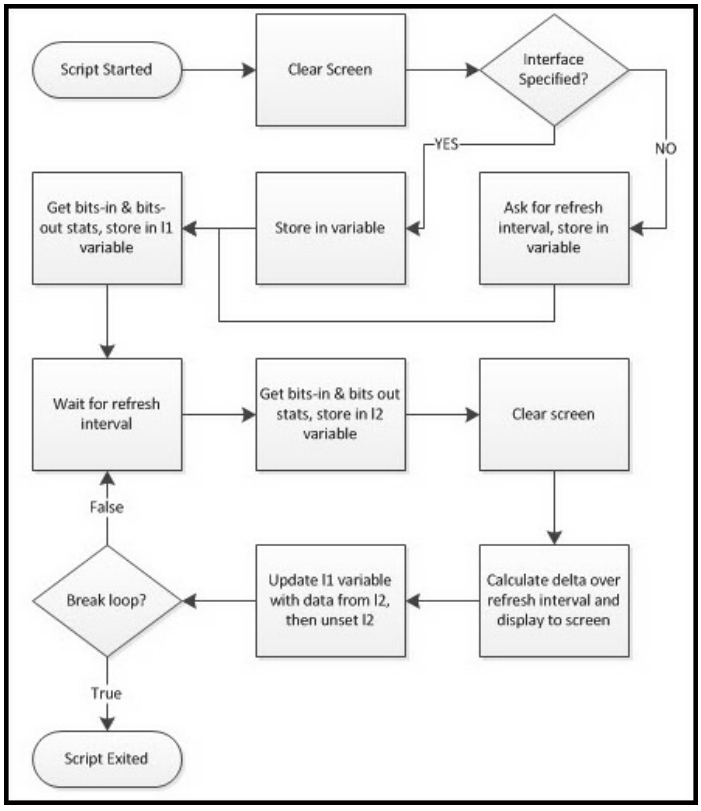
```
% split $x "\n"
net\ interface\ 1.1\ \{ \
{    counters.bits-in 59500356294368} \
{    counters.bits-out 5082163022832} \
{    counters.drops-all 0} \
{    counters.errors-all 0} \
```

```
{    counters.pkts-in 6293231170} \
{    counters.pkts-out 2489470246} \
{    media-active none} \
{    name 1.1} \
{    status up} \} \
{}
% lindex [split $x "\n"] 1
    counters.bits-in 59500356294368
% lindex [split $x "\n"] 2
    counters.bits-out 5082163022832
% lindex [lindex [split $x "\n"] 1] 1
59500356294368
% lindex [lindex [split $x "\n"] 2] 1
5082163022832
```

Now that the data is extracted in proper form, we can move on to the script!

Goals & Workflow

The goals for this script are simple: take the values from counters.bits-in and counters.bits-out from a specified interface and display them at a specified refresh interval. We'll get from goals to a script by first working through some workflow:



The Script

Since we need to get data from the user (interface and interval specifications), let's start with the standard input. We'll use the getFeedback proc below.

```
proc getFeedback { question } {
    puts -nonewline $question
    flush stdout
    return [gets stdin]
}
```

This proc pulls is then used in the initial script setup as shown next.

```
tmsh::clear_screen
if { $tmsh::argc == 1 } {
    set int [getFeedback "Please enter the interface number (ie, 1.1): "]
} else {
    set int [lindex $tmsh::argv 1]
}

set l1 []
set l2 []
set interval [getFeedback "Please enter refresh rate for the stats (in seconds): "]
set delay [expr $interval * 1000]
```

Here we see the screen has been cleared, and then if the only argument in the script initialization is the script itself, then we ask for the interface name. Otherwise, we take the second argument value and set it as the interface name. Then, we initialize the l1 and l2 variables as lists. Finally, we ask for the desired refresh interval and set that delay for the after command use as it's argument is in milliseconds, not seconds. Next, we need to go ahead and take the data and dump it into the l1 variable we initialized:

```
lappend l1 [lindex [lindex [split [tmsh::show net interface $int raw field-fmt] "\n"] 1] 1]
lappend l1 [lindex [lindex [split [tmsh::show net interface $int raw field-fmt] "\n"] 2] 1]
```

It looks a little scary, but this is an exact copy of the structure shown above in the Tcl shell except that we're using the TMSH command output instead of the static "x" variable we used to get the syntax necessary to extract the data. This results in l1 having a list with the bits-in and bits-out values in indexes 0 and 1 respectively. Now, the loop that allows this script to display the bit rate real time.

```
while { true } {
    after $delay
    lappend l2 [lindex [lindex [split [tmsh::show net interface $int raw field-fmt] "\n"] 1] 1]
    lappend l2 [lindex [lindex [split [tmsh::show net interface $int raw field-fmt] "\n"] 2] 1]
    tmsh::clear_screen

    set statsIn [expr ([lindex $l2 0] - [lindex $l1 0]) / $interval]
    set statsOut [expr ([lindex $l2 1] - [lindex $l1 1]) / $interval]

    puts "Interface\t\tInbound (bps)\t\tOutbound (bps)"
    puts "$int\t\t\t$statsIn\t\t\t$statsOut"

    set l1 $l2
    unset l2
}
```

This loop will continue until you break it with a ctrl-c. We start the loop condition with our specified delay, then do with the l2 variable what we did with the l1 variable: take a snapshot of the bits-in and bits-out of the interface. After again clearing the screen, now we take the delta of the new snapshot and the old snapshot, and divide by the interval to get the bits transferred in and out on that interface, per second. Next, we display that to the screen with the puts command. Finally, in order to maintain the latest snapshot for the next interval, we set the l2 data to the l1 variable and unset the l2 variable. And that's it. Not that complicated, right?

Going Forward

This is a very simple throwaway script that needs a lot of work to have "arrived." Error checking, extensibility, etc, are missing, and are all left to the reader to develop for those purposes. This met a very specific troubleshooting need in my environment, and I would be remiss if I didn't share. I'd love to see someone take on error checking, or maybe displaying the bitrates for all interfaces if none is specified, or going a step further, summarizing all interfaces per vlan and showing vlan bitrates. Any takers?

The script in its entirety is here in the TMSH codeshare.

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

©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, express or implied, claimed by F5. CS04-00015 0113