Scripting on Routers

Richard A Steenbergen <ras@nlayer.net> nLayer Communications, Inc.

Why Script on Routers?

- What do we mean by “Scripting on Routers?”
  - Provide network operators with the ability to write custom software which runs on routers, to simplify configurations, react to network events, and/or automate complex tasks.
- Why do we need scripting on routers?
  - Router configurations are complex and often repetitive.
    - Scripts can simplify existing repetitive configurations.
    - Can also enable new features which might otherwise be impossible.
  - Humans frequently make mistakes.
    - Scripts can provide complex error checking to prevent accidents.
  - Humans cost a lot of money to maintain and administer.
    - Scripts can reduce the manpower necessary to run a network.

What About Offline Automation?

- We have existing router management tools today
  - Expect/Perl/etc scripting over CLI Telnet/SSH
  - IETF standardized NetConf (XML based) Protocol
  - Some networks are entirely managed offline
    - With no humans logging into routers, only scripts.
- But most networks are still run the old fashioned way
  - Writing offline tools requires dedicated/experienced staff.
  - It is difficult to pre-plan for every possible configuration.
  - Most networks use a mix of tools + standard CLI configs.
  - Scripting on the router provides the advantages of router management software, while still allowing manual one-offs.

Some Router Scripting Examples

- Example automation of 3 common networking tasks
  - Automated BGP Policy Generation
    - Per-ASN BGP communities and their associated policies.
    - Other Per-ASN policies such as AS-PATH leak filters.
    - All automated and built automatically for every BGP peer.
  - Automated BGP Prefix-Limit Management
    - Auto-tuning prefix-limits which adjust to follow changes in BGP.
    - Without requiring human intervention to maintain.
  - Support case data gathering scripts
    - Automatically compiles and uploads logs/info when opening cases.

Automated BGP Policy Generation

Automated BGP Policy Generation

- Start by defining a “BGP location” macro:
  - Essentially a piece of custom router configuration which operators can maintain, to be used by the script later on.
  - Here we define the values for continent, region, and city codes to be used for BGP Communities for the router.
  - system {
    location {
    apply-macro bgp {
    city 16;
    continent 1;
    region 2;
    …

Automated BGP Policy Generation

- Configure a BGP Peer like normal

```
protocols {
        bgp {
                group PNI {
                        import PRIVPEER-IN;
                        export PRIVPEER-OUT;
                        neighbor 1.2.3.4 {
                                description “Example BGP Peer”;
                                peer-as 1234;
                        }
        }
}
```

Automated BGP Policy Generation

- Commit the configuration, and the script runs.
- The script reads the “location” data from our config:
  - var $location = system/location/apply-macro[bgp];
    var $continent = $location/data[‘continent’]/value;
    var $region = $location/data[‘region’]/value;
    var $city = $location/data[‘city’]/value;
- Calls a function to build policy for every BGP neighbor
  - for-each (protocols/bgp/group/neighbor[peer-as]) {
      call example($asn, $name, $continent, $region, $city);
      …
    }

Automated BGP Policy Generation

- Generate some new configurations defining BGP policies

```
<policy-statement> {
        <name> "AUTO-COMMUNITY-" _ $name _ "-OUT";
        <term> {
            <name> "PREPEND_ONE";
            <from> {
                <community> "MATCH_" _ $name _ "_PREPEND_ONE";
            }
            <then> {
                <as-path-prepend> $local-as;
            }
        }
…
```

Automated BGP Policy Generation

- Generate some new configurations defining BGP communities

```
var $regexp   = "((000)|" _ $reg_continent _ "|" _ $reg_continentregion _ "|" _ $reg_city _ ")$";
call jcs:emit-change($tag = 'transient-change', $dot = $path/policy-options) {
        with $content = {
                <community> {
                        <name> "MATCH_" _ $name _ "_PREPEND_ONE";
                        <members> "^" _ $asn _ ":1" _ $regexp;
                }
                <community> {
                        <name> "MATCH_" _ $name _ "_PREPEND_TWO";
                        <members> "^" _ $asn _ ":2" _ $regexp;
                }
                <community> {
                        <name> "MATCH_" _ $name _ "_PREPEND_THREE";
                        <members> "^" _ $asn _ ":3" _ $regexp;
                }
```

Automated BGP Policy Generation

- Script builds per-ASN communities/policies for:
  - Prepend AS-PATH 1x
  - Prepend AS-PATH 2x
  - Prepend AS-PATH 3x
  - Prepend AS-PATH 4x
  - Set BGP MED to 0
  - Deny export of BGP route
  - Allow export of BGP route (override a broader Deny)
- Also builds generic community tags for the router:
  - Match routes from current continent/region/city
  - Tag route learned in current continent/region/city

Automated BGP Policy Generation

- Dynamically creates community expressions that look like this:

```
community MATCH_3356_PREPEND_ONE members "^3356:1((000)|(010)|(012)|(116))$";
community MATCH_3356_PREPEND_TWO members "^3356:2((000)|(010)|(012)|(116))$";
community MATCH_3356_PREPEND_THREE members "^3356:3((000)|(010)|(012)|(116))$";
community MATCH_3356_PREPEND_FOUR members "^3356:4((000)|(010)|(012)|(116))$";
community MATCH_3356_MED_ZERO members "^3356:5((000)|(010)|(012)|(116))$";
community MATCH_3356_DENY_EXPORT members "^3356:6((000)|(010)|(012)|(116))$";
community MATCH_3356_FORCE_EXPORT members "^3356:9((000)|(010)|(012)|(116))$";
```

- And the policies which reference these expressions

```
 term PREPEND_ONE {
    from community MATCH_3356_PREPEND_ONE;
    then as-path-prepend “1234”;
}
term PREPEND_TWO {
    from community MATCH_3356_PREPEND_TWO;
    then as-path-prepend “1234 1234";
}
```

Automated BGP Policy Generation

- Also useful for building AS-PATH leak filters
  - Define a list of major ASNs you only want to hear “directly”
    - Block any route with one of these reserved ASNs in the AS-PATH if the route didn’t come directly from one of those ASNs.
  - Useful for preventing leaks and suboptimal routing.
    - If I peer directly with AS701, I don’t ever want to accept a route with 701 in the AS-PATH from anyone other than a AS701 neighbor.
  - The same script can build a per-peer AS-PATH filter which blocks every ASN on the list except the ASN of the peer.
  - This can’t be done via RegExp in Cisco or Juniper (or any other major router vendor) today, a per-ASN AS-PATH filter policy is the only way to accomplish this task.
  - This filter is highly effective at blocking accidental leaks.

Automated BGP Policy Generation

- Also useful for building policy frameworks
  - Script scan for the existence of a policy with a standard name (e.g. POLICY-AS####-IN) for every BGP neighbor.
  - If the policy exists, it is automatically linked in to the policy chain as a “subroutine” to existing standardized policies.
  - The standardized policy enforces certain requirements for the BGP neighbor, such as the BGP Community tag.
  - While also allowing operations staff to tweak the route.
  - Allow your operations staff to tweak a local-pref or a MED without ever being able to accidentally leak the route.
  - Simplifies BGP configuration, policies linked automatically.

Automated BGP Policy Generation

- Insert the newly created policies into the import/export policy chains

```
for-each (protocols/bgp/group/neighbor[peer-as]) {
        var $import = jcs:first-of(import, ../import, ../../import);
        var $export = jcs:first-of(export, ../export, ../../export);
        var $in_first = $import[position() = 1];
        var $out_first = $export[position() = 1];

        call jcs:emit-change($tag = 'transient-change') {
            with $content = {
                <import> $import;
                <import insert="after" name=$in_first> "AUTOCOMM-" _ peer-as _ "-IN";
                <export> $export;
                <export insert="after" name=$out_first> "AUTOCOMM-“ _ peer-as _ "-OUT";
            }
        }
}
```

Automated BGP Policy Generation

- Script Results
  - Script automatically generates and maintains 44 lines of new router configuration for every configured BGP peer.
  - Script automatically build many of the BGP community definitions for the router.
  - Script automatically links all the new policies together correctly, avoiding rote human effort and potential mistakes.
- The Net Effect
  - For routers with hundreds of BGP peers, thousands of lines of configuration are automated, the user visible config is simplified.
  - Enables new features (per-ASN communities, leak filters, policy framework, etc) that most networks don’t implement today.

Automated BGP Prefix-Limit Management

BGP Prefix Limits

- Operators use BGP prefix-limits as policy safety nets
  - If a BGP neighbor sends more prefixes than we believe is normal, drop the BGP session for a certain period of time.
  - Somewhat effective at protecting against the propagation of major leaks, and a commonly used tool for most peers.
- But they are somewhat difficult to maintain
  - The concept of “normal” is always evolving, as networks grow, shrink, or otherwise change their announcements.
  - Stale prefix-limit configurations are probably responsible for thousands of peering outages every year.
  - But keeping the limits set at sensible values is hard work.

Use Router Scripting to Automate Limits

- How do you determine a sensible prefix limit?
  - Typically based on the number of current prefixes.
  - Plus some percentage extra for growth
  - Plus some fixed number to handle small prefix counts.
  - Example: (PfxCnt*1.25)+500
- But you also want to react to changes slowly
  - Don’t slash your prefix-limit because the peer happens to be announcing 0 prefixes due to an outage one night.
  - Use a weighted moving average to adjust the limit slowly over time, towards the newly computed value.

Use Router Scripting to Automate Limits

- Write a script to update the prefix-limit accordingly.
- Have it run automatically on the router every night.
- Allow a manual run to tune the limits if necessary.

```
var $config = jcs:invoke($get-config)/configuration/protocols/bgp;
var $neighbors = jcs:invoke('get-bgp-summary-information');

for-each ($config/group/neighbor[peer-as]) {
    var $address = name;
    var $neighbor = $neighbors/bgp-peer[peer-address == $address];
    var $pfxlimit = apply-macro[name == 'prefix-limit']/data[name == 'inet'];
    var $pfxrecv = $neighbor/bgp-rib[name == 'inet.0']/received-prefix-count;
    var $rcvlimit = ceiling((($pfxrecv * 1.25) + 500) div 500) * 500;
    …
```

Automating Support Cases

Opening Vendor Support Cases

- We all find bugs or need to open support cases
  - Many times gathering the support information and uploading them to the Vendor can be long and painful.
  - Support information, current configuration, scripts, log files, on the active and backup management cards, etc.
- Solution: Automate it with a script
  - Gather the most common log files and support components, and automatically upload them to the vendor FTP site.

Gather and Upload the Data

```
var $dest = jcs:first-of($destination, "ftp://ftp.juniper.net/pub/incoming/");
var $support = jcs:invoke('get-support-information');
var $config  = jcs:invoke('get-configuration');
call upload($file = $re _ ":/tmp/support", $dest = $dest _ $case);
call upload($file = $re _ ":/tmp/configuration", $dest = $dest _ $case);
call upload($file = $re _ ":/var/log/messages", $dest = $dest _ $case);
call upload($file = $re _ ":/var/log/chassisd", $dest = $dest _ $case);
…
var $filename = jcs:regex("[^\/]+$", $file)[1];
var $destfile =  $hostname _ "." _ $filename;
var $copy_upload = {
        <file-copy> {
            <source> $tmpfile;
            <destination> $dest _ "/" _ $destfile;
        }
    }
<output> "Uploading " _ $filename _ " to " _ $dest;
jcs:invoke($copy_upload);
```

How Effective Is It?

- Router configuration size reduced by 62%

ras@randomrouter> show configuration | count

Count: 2587 lines

ras@randomrouter> show configuration | display commit-scripts | count

Count: 6742 lines

Send questions, comments, complaints to:

Richard A Steenbergen ras@nlayer.net