JunOS BGP

Pedro Roque Marques
roque@juniper.net
JunOS

- **Routing Engine (RE)**
  - Unix kernel
  - Daemons
    - Routing (rpd)
    - Interface management
    - Configuration management
    - SNMP
    - CLI

- **Packet Forwarding Engine (PFE)**
  - PPC microprocessor (for control)
  - Forwarding ASIC
Routing Protocol Daemon

- IP/MPLS routing and signaling.
- Shared data structures:
  - Multiple routing tables
    - inet.0 - IP routing table
    - inet.3 - MPLS routing table
    - routing-instance tables
- Forwarding information installed in the kernel.
- Kernel distributes to control CPUs on line cards.
Configuration

- **Database model**
  - Commit/rollback.
  - No actions in inconsistent state.

- **Without a database model:**
  - “no access-list 101”
  - Followed by new access-list configuration.
  - Any routing updates received are processed according to the current state.
  - Race between operator/script and events on the network.
“commit” processing

- Subsystems calculate differences between previous and new state.
- Reevaluate received/advertised routing information.
- Can lead to high CPU utilization/poor response time to new events.
- Improvements:
  - Skip database objects that were not modified.
  - Incremental changes (5.2, 5.3, 5.5)
- Influential in the design of the system.
Adjacency maintenance

- RPD cooperative multitasking nature:
  - “scheduler slips”.
  - Specially while processing reconfiguration (before configuration change improvements).
- Periodic packet management (ppmd)
- Protocol hellos (ospf, isis).
- “real time” component of routing.
- In JunOS R5.3
JunOS BGP

- **Receiving Updates**
  - Apply inbound policy (import).
  - Keep copy of original attributes.

- **Route selection**
  - Centralized (all routes from all protocols).

- **Update generation**
  - “export” policy applied on selected route.
  - Per-group copy of advertised attributes.

---

**Diagram**

1. **Import Updates**
   - RIB-In

2. **Route Selection**
   - Loc-RIB

3. **Export updates**
   - RIB-Out
Receiving UPDATEs

- Display received attributes:
  - “show route receive-protocol bgp <neighbor> all”
Policy language

- Sequence of terms with \(<\text{match}, \text{action}\)> pairs.
- Policies can be used as sub-routines.
- Policies can be chained.

```plaintext
policy-options {
  policy-statement example {
    term one {
      from policy (a && b);
      then accept;
    }
    term two {
      then next-policy;
    }
    then reject;
  }
}
```
By default, reject routes with:

- As-path loop.
- Cluster-list loop.
- Invalid 3rd party eBGP next-hop (non-connected).
- neighbor “keep all” option disables this behavior.

In case of a configuration change that modifies the neighbor “import” policy:

- New route attributes are automatically calculated.
- No need to request REFRESH to N neighbors; local processing.
Route selection

- Includes routes from all protocols.
- E.g.: static (higher admin preference) and BGP route present. Inactive BGP route is not advertised.
- May override this behavior with "advertise-inactive" knob.
- BGP path selection depends on IGP metrics.
- Triggered by receiving a routing update or resolution metric change.
Generating UPDATEs

- Best route for a given prefix is selected.
- Per peer-group processing of routes.
- Default export policy: accept all BGP routes.
- Generate advertisement metrics (values for BGP attributes).
- Compare with previously advertised value.
Memory usage: number of groups.

Group: consistent export policy.

Groups will be split when export parameters differ. Configuration is a template.

“show bgp group”.

“show route advertising-protocol bgp <neighbor>”.

- Displays what the router has advertised.
- Equiv. commands in other implementations rerun their export processing and calculate what would be advertised w/ current state.
Send process

- Encode metrics calculated via export policy.
- NLRI for which “export” is pending is delayed.
- Replicates UPDATE to members of the group.
- Efficient packing of NLRI with same set of attributes.

```
 neighbor A
     ▸ ATTR1
     ▸ Prefix 1
     ▸ ADVRT1
     ▸ Prefix 1
     ▸ Prefix 2
     ▸ BGP UPDATE
     ▸ neighbor A
   ▸ neighbor B
   ▸ ...  
   ▸ neighbor N
```
Send process

- Send may be synchronous or asynchronous in relation to “export” stage.
- Differences in load and latency between members of group.
- When a receiver blocks its TCP socket it gets “out-of-sync”.
- Generate UPDATEs for subsets of “out-of-sync” peers at same stage in queue (R5.4).
- “show bgp summary” – “OutQ” field
**Route Resolution**

![Diagram showing route resolution with prefixes and protocols]

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Protocol</th>
<th>Next-hop</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.1/24</td>
<td>BGP</td>
<td>20.1.1.1</td>
<td>-</td>
</tr>
<tr>
<td>20.1.1.1/32</td>
<td>ISIS</td>
<td>&lt;interface A&gt;</td>
<td>20</td>
</tr>
</tbody>
</table>

- **BGP route selection depends on IGP metric**
IGP change

- Immediate change for all IP prefixes dependent on C.
External route change

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Protocol</th>
<th>Next-hop</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.1/ 24</td>
<td>BGP</td>
<td>20.1.1.1</td>
<td>-</td>
</tr>
<tr>
<td>10.0.1/ 24</td>
<td>BGP</td>
<td>20.2.2.2</td>
<td>-</td>
</tr>
<tr>
<td>20.1.1.1/ 32</td>
<td>ISIS</td>
<td>&lt;interface B&gt;</td>
<td>30</td>
</tr>
<tr>
<td>20.2.2.2/ 32</td>
<td>ISIS</td>
<td>&lt;interface B&gt;</td>
<td>25</td>
</tr>
</tbody>
</table>
Route Resolver

- Change in IGP metric results on running route selection on dependent routes.
- Resolution independent of protocol:
  - Same mechanism for IP and MPLS
- Recursion loop avoidance.
- Forwarding table update:
  - 1 update to PFE for change in IGP next-hop
  - No recalculation on PFE.
  - Fast convergence.
- “show route resolution”
JunOS BGP Characteristics

◆ Changes in “import” or “export” policy.
  ❖ Router updates its states so that it is consistent with the new configuration.
  ❖ Differential updates are sent for export changes.

◆ BGP always knows what it actually advertised to a given peer/group. No duplicate advertisements.

◆ Not optimized for small memory footprint.

◆ No periodic processing.
Address family support

- The “What about IPv6?” question.
- NLRI-types supported: inet, inet6, inet-labeled, inet6-labeled, inet-vpn, inet6-vpn, l2vpn...
- Validation of received attributes and selection of advertisement attributes is a per NLRI-type function.
- BGP as a database exchange protocol:
  - Flow control between peers.
  - Flooding of routing information (loop avoidance).
L3VPNs

- PE routes are received into a separate BGP table (bgp.l3vpn.0).
- Route selection performed in Red/ Green instance (VRF) table.
- Difference from inet routes: add/ strip RDs.
Scalability

- Growth of Internet routing-table prefixes:
  - When do we run out of memory?
  - When do we run out of CPU?
- Memory consumption is a function of the number of distinct paths (!= prefixes).
- Juniper currently maxes out at 2GB RAM and about 1M prefixes in forwarding engines.
- CPU maybe more of a concern:
  - In terms of the number of distinct change sets that occur.
  - Some overhead in terms of per prefix processing.
- L3VPN (2547) RRs “pushing the envelope” in terms of number of prefixes and events.
Configuration tips

- BGP configuration is hierarchical.
- Less lines makes it easier to read/understand.
- "peer-as" unnecessary in internal groups.
- "local-as" unnecessary if same as autonomous-system setting.

```bash
protocols bgp {
    /* top level */
    group <name> {
        type [internal | external];
        export <policy>;
        neighbor <address> {
            peer-as <as#>;
        }
    }
}
```
Securing BGP

- Filter own prefix at edges to avoid spoofed sources.
- Firewall filter can be applied to packets addressed to the router.
- Automatic expansion of configured BGP peers.

```
[edit policy-options prefix-list foo]
# set apply-path "protocols bgp group <*> neighbor <*>"

[edit firewall family inet filter bar]
# set term 1 from source-prefix-list foo
# set term 1 from destination-port bgp
# set term 1 then accept
```
Unicast Reverse Path Check

- IP source address anti-spoofing
- Verify that the source was received from an interface or set of interfaces that match reverse path.
- 3 modes available:
  - Best path
  - All feasible paths
  - Any feasible path
- Feasible-path option can work with multi-homing if peer/customer advertises all prefixes on all connections.
Feasible paths

Scenario:
- Provider accepts MEDs for customer routes.
- Preferred exit point for an external network is GW-1.
- Accepting source from a feasible path allows SP to perform anti-spoofing validation.
Thank You

http://www.juniper.net