IETF 92
NETMOD WG

Consistent Modeling of Operational State Data in YANG

OpenConfig network operator working group
www.openconfig.net

draft-openconfig-netmod-opstate-00

Rob Shakir (BT), Anees Shaikh, Marcus Hines (Google)

Operational state is critical for management

- network operational state is required for turnup, troubleshooting, traffic management, network planning, ...
- operational state access is much more frequent than configuration
  - think streaming telemetry and pub-sub, not SNMP polling
- state data is generally higher volume than configuration data

Types of operational state data

- derived, negotiated, set by a protocol, etc. (negotiated BGP hold-time)
- operational state data for counters or statistics (interface counters)
- operational state data representing intended configuration (actual vs. configured)

Limited attention paid to modeling operational state

Clear benefits from using YANG to model both configuration and operational state in the same data model

but there are limitations ...

- “The primary focus of YANG is configuration data.” [RFC 6244]
  - (NETCONF doesn’t help by blurring distinctions in types of state data)
- NETCONF-centric view assumes transactional, synchronous behavior
- no common conventions for structuring operational state data
  - further exacerbated when composing models
- most YANG models being developed are not paying attention to state data

Requirements for modeling operational state data

- intended configuration as part of operational state
  - intended and actual configuration values can be different
- support for both transactional, synchronous and distributed, asynchronous management systems
  - configuration changes may not be applied immediately or atomically
- separate and retrieve configuration and operational state data independently
- retrieve only the state corresponding to derived values and statistics
  - avoid duplicate data transactional / synchronous systems
- consistent schema locations for configuration and corresponding operational state data
  - correlating configuration and state data should be simple
  - read actual and intended configuration simultaneously to compare

Proposed structure for configuration and state

```
+--rw example container
    +--rw config
    |  +--rw conf-1?    empty
    |  +--rw conf-2?    string
    +--ro state
       +--ro conf-1?      empty
       +--ro conf-2?      string
       +--ro state-1?     boolean
       +--ro state-2?     string
       +--ro counters
          +--ro counter-1?   uint32
          +--ro counter-2?   uint64
```

configuration data

intended configuration state data

derived state data

counter state data

tagged `operational: true`

more complete examples in draft-openconfig-netmod-opstate-00

Some pros and cons

Advantages

- addresses all of the operational requirements for a variety of scenarios (some cases require more thought)
- not affected by model composition -- location of configuration and corresponding state in the path are known with no external context needed
- supportable by existing implementations (suboptimally by NETCONF)

Drawbacks

- writing models requires some additional work
- requires modelers to follow a convention
- extra containers in the path required to achieve model symmetry

trade off additional modeling effort for simplified consumption of operational state data

Topics for discussion

- Modeling design patterns
  - basic container groupings, handling lists, selective sharing of state data, state with no corresponding config
- YANG language implications
  - list keys
  - rw data in ro containers
  - extensions to mark derived state vs. intended config state

discussed in the draft

- Alternative approaches
  - use datastores / RPCs to distinguish the data instead of the schema
  - ‘flatten’ the schema, requiring only a config or only state container
  - http://www.openconfig.net/file-cabinet/Modelling-Operational-State.pdf alternative approaches considered.

Additional material

Example

```
+--rw interfaces
   +--rw interface* [name]
      +--rw name        -> ../config/name
      +--rw config                                  /interfaces/interface[name=ifName]/aggregation/config/...
      |    ...
      +--ro state
      |  | ...
      |  +--ro counters                             /interfaces/interface[name=ifName]/aggregation/state/...
      |     +--ro discontinuity-time
      +--rw aggregation!
         +--rw config
         |  +--rw lag-type?    aggregation-type
         |  +--rw min-links?   uint16
         +--ro state
            |  +--ro lag-type?    aggregation-type
            |  +--ro min-links?   uint16
            |  +--ro members*     ocif:interface-ref
            +--rw lacp!
               +--rw config          /interfaces/interface[name=ifName]/aggregation/lacp/config/...
               |  +--rw interval?   lacp-period-type
               +--rw members* [interface]
               |  +--rw interface    ocif:interface-ref
               |  +--ro state        /interfaces/interface[name=ifName]/aggregation/lacp/
               |     +--ro activity?          lacp-activity-type   members[name=ifName]/state/...
               |     +--ro timeout?           lacp-timeout-type
```