Observations on Modelling Configuration and State in YANG.

{robjs, aashaikh}@google.com

Background.

- Since ~Summer 2014, OpenConfig has:
  - Focused on covering a “operationally viable” subset of the configuration and state of routing, switching and optical devices.
  - Published an ever-growing set of YANG models.
  - Focused on implementations by network equipment vendors, after reviews with network operators.

- Asked to give some feedback on our experience.
  - Not going to talk about YANG language features here - have raised specific concerns.
  - **DISCLAIMER: We are not asking for the IETF to do anything about our observations - we’re just sharing knowledge.**

Some more details: what has OpenConfig built?

- 72 YANG modules and supporting developer infrastructure.
  - Coverage for L2 switches, IP routers, IP/MPLS LER/LSRs.
  - Transport devices - amplifier, ROADM (“wavelength router”), terminal devices.

- YANG tooling.
  - A YANG compiler (goyang)
  - Python & Go language binding generators with validation backends.
  - Plugins for documentation, path extraction, generating alternate schema representations.

- Configuration and state manipulation protocol, and tooling.
  - gNMI specification and proto.
  - Reference collector implementation.

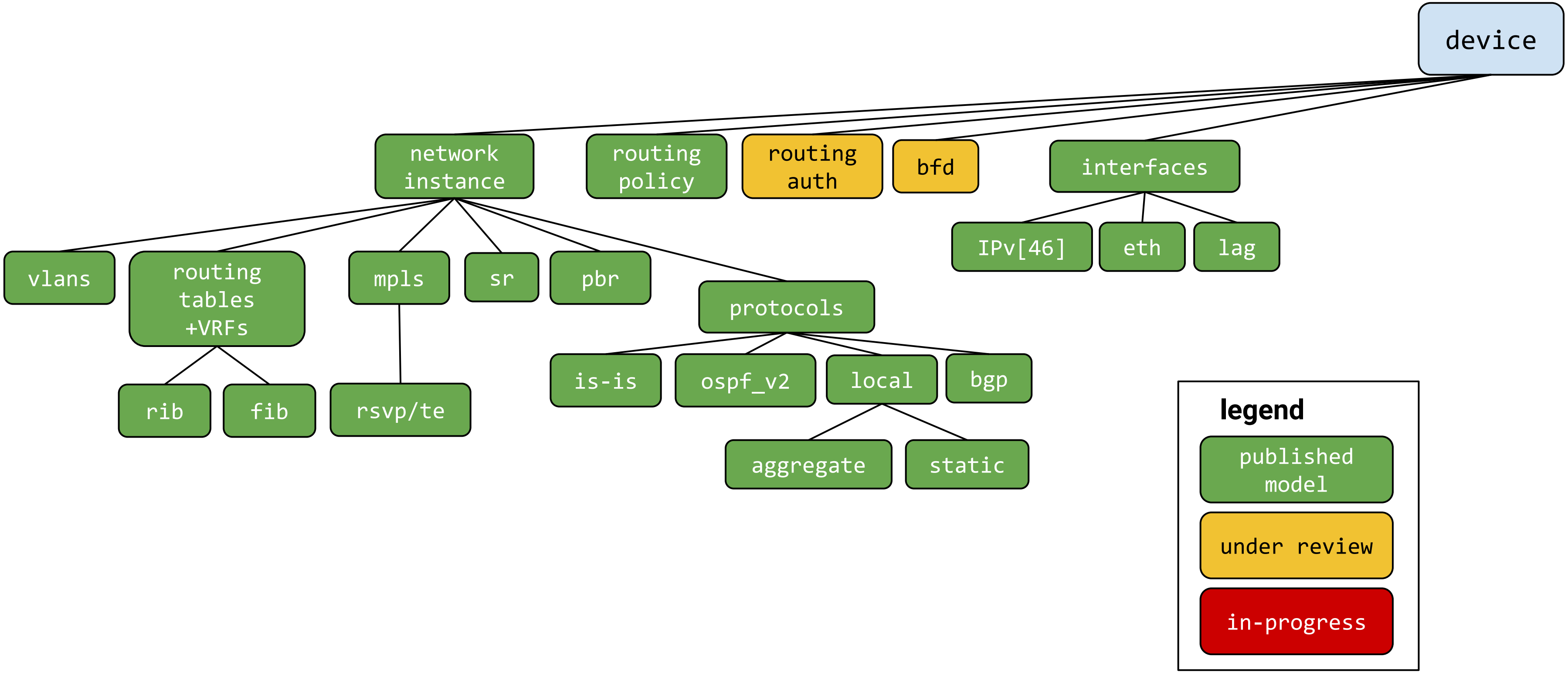
Some more details: implementations.

- A number of major vendors have shipping code that supports OpenConfig models.
    - We directly interact with >5 vendors, based on OpenConfig member customer demand on issues mapping to their underlying schemas.
    - Grateful to these folks for their input - *launch-and-iterate* approach to getting usable models.

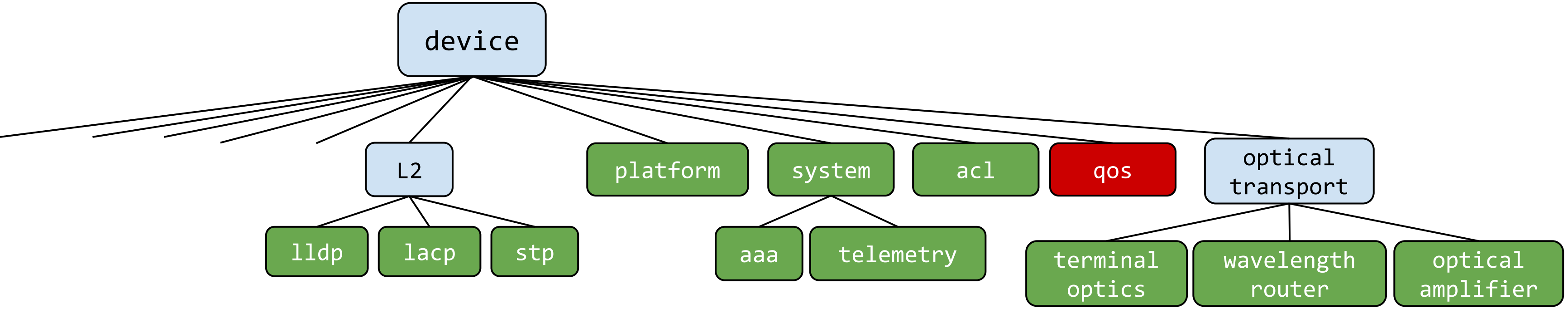
- Models are driving multiple operator’s NMS stacks.
    - A standard representation for telemetry variables across multiple platforms.
    - Vendor neutral configuration specification language.

Per feedback at IETF92, aiming to inform discussion with running code.

The OpenConfig Model Landscape (I)



The OpenConfig Model Landscape (II)



device

L2

lldp

lacp

stp

platform

system

aaa

telemetry

acl

qos

optical transport

terminal optics

wavelength router

optical amplifier

legend

- 🟩 published model
- 🟨 under review
- 🟥 in-progress

Some key observations.

- Folks **don’t care that you’re using YANG**...
  - People interacting with network devices want to **do something**, not care about the modelling language.
  - Our philosophy is to try and ensure that we don’t have to teach people about YANG, unless they’re actually writing schema modules.

- **Consistency is key**...
  - If you have to explain that “LLDP works like this, but LACP works like this”, then you’ve already failed.
  - Do not want to trade the complexity of heterogeneous vendor configuration formats for that of inconsistent data models.
  - We’ll use the word **consistent** a lot!

Everything is State.

- We can’t just design models based around configuration data.
    - **How** and **where** to model operational variables is *critical.*
    - We think of things in terms of **intended**, **applied** and **derived** state.
    - Still no consensus around **opstate** in the IETF (we tried…).

- Consistency around **where** a user finds state variables is important.
    - If this needs explaining per model, we’ve failed.

- Consideration of telemetry is needed throughout models.
    - e.g., how do we send an efficient `delete` update for a keyless list?
    - Are there ways we can design the models to allow for related variables to be transmitted together?
    - How do we annotate the schema to indicate different data types?

Most difficult models: unifying other models.

- Case in point: *openconfig-network-instance.*
    - Model that unifies a number of entities within OpenConfig.
    - Protocols, AFTs, tables (RIBs).
    - Allows multi-tenancy of a network element (VRFs, VSIs...).

- Needs to have a basic set of functionality which is well understood by operators.
    - e.g., how protocols redistribute routes between each other.
    - Minimum viable set is critical - understanding operational requirements.

- Non-trivial to map to underlying vendor implementations consistently.
    - We have done work to map OC-NI to 4 different vendors' CLI implementation.

It’s not about the “best” data model.

- OpenConfig tries to concentrate on *operationally usable* models.
    - Try and think in terms of **how features are used** rather than **how they look on the wire** or **how they are specified**.

- The other factor we optimise for is implementability.
    - Some duplication exists for compatibility reasons (more granular support/less granular support).
    - Some “mode” flags to support different implementations.
    - Balance mapping complexity across implementations.

- Don’t discover some issues until review/implementation time.
    - Iteration is required in the models - private and public engagement.
    - Incompatible with standardise then implement.

It’s not about the most complete data model.

- Implementation and review effort is leaf-by-leaf.
  - This is how implementors (vendors, internal operator code) generally engage with the models that we publish.
  - Obvious: the more leaves, the more review required, the more code to be written.

- Implementation code is for mapping configuration or state data, or to add internal instrumentation for telemetry.

- Observation: Biasing towards operationally used features is key.
  - Catalogue ‘`feature-bundles`’ allow operators/vendors to specify their unit of compliance.
  - Avoids an ocean-boiling exercise.

Versioning is more complex than `revision`.

- The YANG `revision` semantics don’t easily map to real world iteration.
  - There will be some backwards incompatible changes.
  - Revision gives zero information as to what the type of change is.
  - Seeing others (not just OC) use some alternate versioning.

- Versioning gets harder for combinations of models.
  - What works with what? What functionality can be supported with a particular set of models.

- OpenConfig approach:
  - Semantic versioning (How did this model change?) - `openconfig-version`
  - Model cross-products (What models work together?) - `release-bundle`.
  - Compliance units per operator/vendor (What is supported where?) - `feature-bundle`.

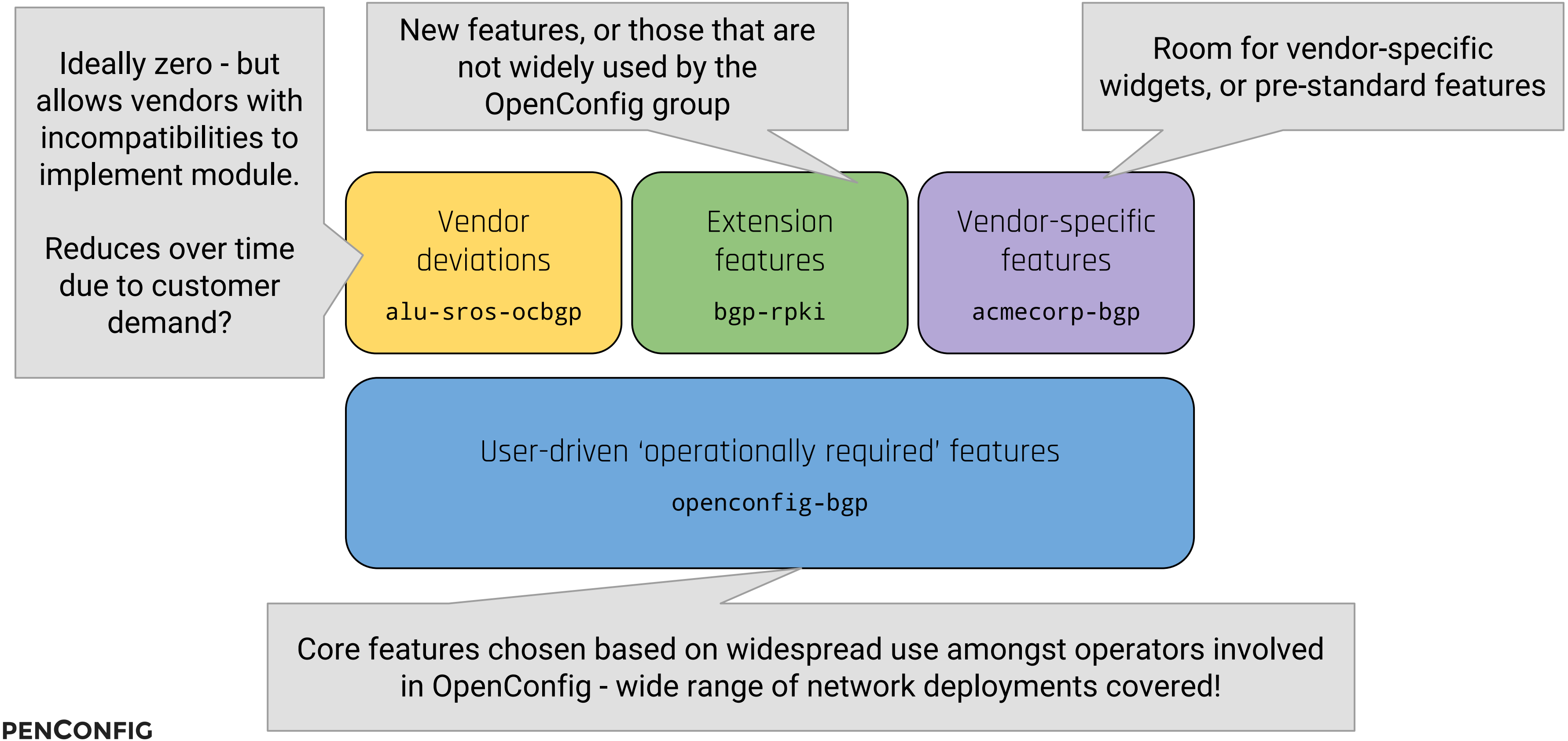
Constraining Language Feature Complexity.

- There are lots of degrees of freedom in YANG.
  - Some of the functions overlap - e.g., `choice/case` vs. `when`.

- Code generation has to consider how to map these into usable artifacts.
  - Unions of unions of unions….
  - Unions of multiple enumerations.
  - Defaults that apply to one of N different member types.
  - How to represent presence within a data structure.

- Possible to use all these features - but increases number of bugs in code generation, and effort for implementation.
  - Majority of new features are new YANG combinations of features.
  - Most bugs relate to untested combinations (testing all combinations is not tractable).

The approach to extensibility matters.



Ideally zero - but allows vendors with incompatibilities to implement module.

Reduces over time due to customer demand?

New features, or those that are not widely used by the OpenConfig group

Room for vendor-specific widgets, or pre-standard features

Vendor deviations
`alu-sros-ocbgp`

Extension features
`bgp-rpki`

Vendor-specific features
`acmecorp-bgp`

User-driven ‘operationally required’ features
`openconfig-bgp`

Core features chosen based on widespread use amongst operators involved in OpenConfig - wide range of network deployments covered!

OpenConfig & the IETF.

- **Aim to continue to engage in discussions around modelling.**
  - Not really clear where we should for this (rtgwg? netmod? rt-yang-arch-dt?)
  - Comments on implementation experience seem to be lacking in the IETF.
    - Bias towards running code?

- **Aim to progress models that we have already published.**
  - BGP and policy models are in the IETF today.

- **Observe:**
  - Conclusion to opstate. Solution that is decided on, and implementations.
  - Approach taken to implementability for models in the IETF.
  - Potential fixes for usability issues in future YANG versions (`map`? `posix-regexp`? `leafref` between `config true` and `false`?)

Backup

Must consider implementation complexity.

- And this **MUST** be for **both** for equipment and NMS vendors.

- Example: Regular expressions - w3c standard is not widely used/supported amongst users.
  - Developer needs to understand a new regexp format.
  - Limited existing tools allow you to test against these regexps.

- Example: Lists with keys are not a common data structure.
  - Rather: `dict`, `HashMap`, `Map`.

- These kind of issues result in complexity of implementation - negatively impacts adoption of models.