Network Topology Model
draft-ietf-i2rs-yang-network-topo-12.txt

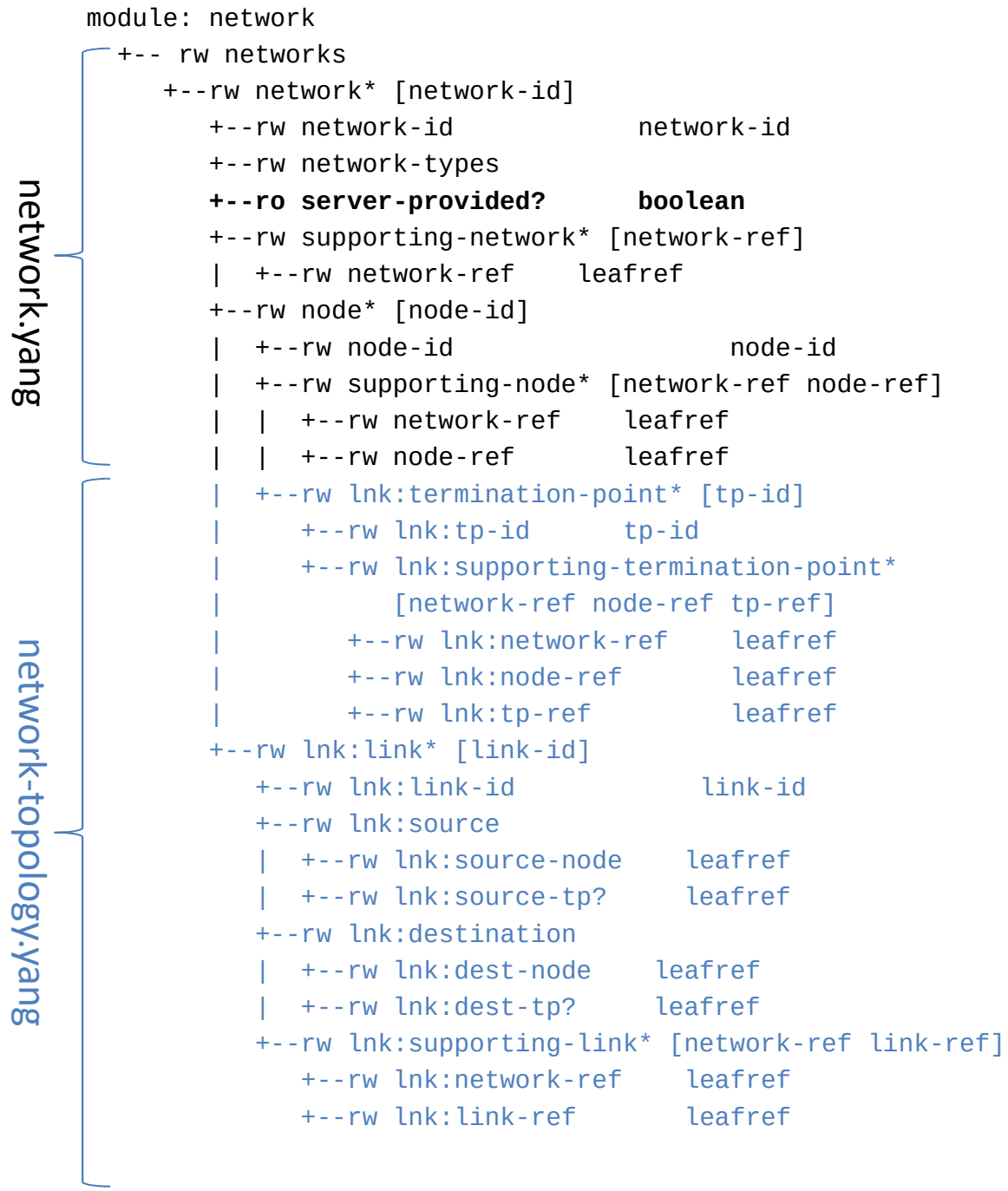
IETF 98, Chicago, 29 March 2017

Alexander Clemm (Huawei), Jan Medved (Cisco),
Robert Varga (Pantheon), Nitin Bahadur (Bracket Computing),
Hari Ananthakrishnan (Packet Design), Xufeng Liu (Ericsson)

Updates

- Went through WGLC yet here we are back again
- Went through six updates (from -06 to -12),
  e.g. with regards to security considerations and use case appendix
- During YANG doctor review, flag was thrown regarding distinction between topologies that are configured vs. topologies that are discovered from the network (“server-provided”)
- Subteam spent several iterations discussing the proper solution
  - Kent Watsen (YANG doctor shepherd), Vishnu Pavan Beeram, authors
  - Susan Hares (document shepherd), Alia Atlas
  - Revisited requirements, collected use cases, documented alternatives
  - Document will be updated as we converge on a consensus

Model Recap



- Express horizontal relationships:
  nodes – tps – links
- Express vertical relationships: layering
- Express various constraints:
  - Supporting nodes/links/tps must be part of supporting (underlay) topo
  - A supporting link must be terminated by a supporting tp on a supporting node
  - Etc
- Base model for more specific topologies that augment this model, e.g. L2, L3, service, …

So, what's the issue

- Some topologies are discovered, others are configured
- E.g. overlays / underlays
- Account for both possibilities in the model while still capturing semantic constraints
- Original solution (still captured in model):
  - Include leaf "server-provided" with each topology that indicates owner/who populated
  - Presence indicates populated by topology discovery app (that coresides on device)
  - Advantages: simple model, current implementations
  - Drawback: "server-provided" data reminiscent of state
    (even if provided by "client" that coresides on server, not unlike other competing-clients scenarios)
    - Locking
    - Backup/Restore will have restored data immediately overwritten
- Various other solutions considered

Tree split option (option 1)

```
module foo {
     container nodes {
        config true;
        list node {
           key "name";
           leaf name { type string; }
           leaf dependency {
              type leafref {
                path "../node/name"
                require-instance false;
                description
               "In the case when a configured node (i.e. in the running DS)
                has a dependency on a node that is not configured, the system
                may try to resolve the dependency as operational state data
                (i.e. under the /opstate-nodes tree).  As operational state
                data may have a lifecycle independent of configuration, there
                is no guarantee that the opstate data will exist.  Therefore,
                application of the configuration node is conditional, resulting
                in an effect much like pre-provisioning interfaces in RFC 7223.";
           }   }
           uses node-attributes;
     }  }
     container opstate-nodes {
        config false;
        list node {
           key "name";
           leaf name { type string; }
           leaf dependency {
              type leafref {
                path "../node/name"
                require-instance false;
           }  }
           uses node-attributes;
 }   }  }
```

Tree split option (option 1)

```
module foo {
     container nodes {
        config true;
        list node {
           key "name";
           leaf name { type string; }
           leaf dependency {
     }  }
     container opstate-nodes {
        config false;
        list node {
           key "name";
           leaf name { type string; }
           leaf dependency {
              type leafref {
                path "../node/name"
                require-instance false;
           }  }
           uses node-attributes;
 }   }  }
```

Both trees will mirror each other

- Equivalent nodes in each
  (not stats in one, config params in the other)
- Augmentation needs to target both trees

Use “grouping” and “uses” to reuse definitions

- Mitigate augmentation complexity through augmentation best practices –
  use grouping/uses to avoid having to augment multiple target nodes with same attributes

Underlay references are “require-instance false”

- State branch object instantiated only when target true

Metadata Option (option 2)

2a: specific to topology
2b: generic, applicable beyond topology

```
module foo {
     import ietf-Netconf {prefix nc;}
     import ietf-yang-metadata {prefix md;}
     md:annotation server-provided {
        type boolean;
     }
     container nodes {
        config true;
        list node {
           key "name";
           leaf name { type string; }
           leaf dependency {
              type leafref {
                path "../node/name"'
    }   }  }  }
    augment /nc:get-config/nc:input {
        leaf with-server-provided {type Boolean;}
}   }
```

- Compare “with defaults” option
- Flag is used to indicate whether to return all data, or configured data only

More alternatives

Shared on the list:

- Option 1: Separate config true and and false trees
- Option 2: Metadata annotation + get-config flag extension for data retrieval

Other flavors considered

- Option 3: Config true (drop “server-provided” leaf)
  - Rely on NACM to withhold authorization to modify server-provided topology layers
  - Eventual migration to revised datastores solution to provide server-provided distinction
- Option 4: Make entire model config false and use RPCs
  - Not very YANG-ish model – replace a model with RPCs
- Option 5: Wait for revised-datastores solution
- Config true (drop “server-provided” leaf)
  - Like option 3: basically, the current model, with “server-provided” leaf dropped
  - Ruled out due to concern that this will hold us back for years (as well as dependent modules)

Per Netmod meeting, revised datastores is close to completion (2-3 months)

- In this case, option 5 suddenly become a lot more attractive….

Recommendation

- Recommendation prior to IETF 98: metadata (option 2A)
  - Easiest and most straightforward to accommodate e.g. by TEAS
  - Avoids tree split, holistic retrieval of topology data
  - Tree split option would have been possible as well, but model complexity a concern
- Recommendation since yesterday: Revised Datastores (option 5)
  - Ruled out initially due to uncertain timeline; having to wait for years not an option
  - Promises to get through the process shortly (2-3 months)
  - Recommendation for new modules to follow
  - Least disruptive with regards to current model
- Implies the following next steps for the draft
  - Update the model (basically, drop server-provided leaf)
  - Add snippets that explain how revised datastores will address the configurable overlay/auto-populated underlay issue
  - Update other models accordingly (e.g. L3 topolog draft-ietf-i2rs-yang-l3-topology)

Is this agreeable to the Working Group? Anything we have missed?

Thank you