RSTP/MSTP YANG updates

Details in …/docs2024/dy-seaman-proposed-yang-module-updates-0906-v03.pdf and the modules themselves …/dy-drafts/d2/dy-yang-modules@2024-08-21/.

This presentation at …/docs2024/dy-seaman-proposed-yang-ppt-0924-v00.pdf

The devil is always in the detail, this presentation is no substitute. It highlights some items, particularly where I am uncertain about YANG best practice.

- Conditionally present/relevant leaf
- Configuration Data — type binary
- Implementation capabilities
- Raw data vs lexical representation

Mick Seaman
mickseaman@gmail.com

Conditionally present/relevant leaf

The Root Port (for the CIST and for each MSTI) is now identified by an if:intreface-ref, as scanning all ports looking for one with a Port Role of Root Port is tedious, particularly for a high port count Bridge and a human user.

There is no Root Port if the Bridge is, itself, the Root Bridge for the relevant tree.

If there was a list or much associated data I would have used a presence container. That's not necessarily the right answer for something dynamic (?), and seems cumbersome for a single leaf. The MSTP module[1] now uses:

```
leaf root-port {
  type union {
    type if:interface-ref;
    type empty;
  }
  …
```

This union was not legal in YANG 1, but is OK in YANG 1.1.

1. The RSTP module needs to be updated if this is OK.

Configuration Data — type binary

The MST Configuration Identifier's Configuration Digest (16 octets) uses type binary. It has length 16, which is easily understood and correct,[1] but the lexical representation is the base64 encoding scheme[2] (Section 4 of RFC 4648) which may give us pause as 13.8 and Table 13-2 (Sample Configuration Digest Signature Keys) use simple hex.

We OK with that? Should we provide the values for the Table 13-2 common cases in description text, or just leave it an exercise for the reader?

All VIDs map to the CIST, no VID mapped to any MSTI
0xAC36177F50283CD4B83821D8AB26DE62
Base 64: rDYXf1AoPNS4OCHYqybeYg==

All VIDs map to MSTID 1
0xE13A80F11ED0856ACD4EE3476941C73B
Base64: 4TqA8R7QhWrNTuNHaUHHOw==

Every VID maps to the MSTID Base64equal to (VID modulo 32) + 1
0x9D145C267DBE9FB5D893441BE3BA08CE
Base64: nRRcJn2+n7XYk0Qb47oIzg==

1. As per 9.8.1 of RFC 7950.
2. In which each printable character represents 6 bits, and special processing is applied if the encoded data is not a multiple of 24 bits (as is the case with our Configuration Digest). The compact encoding is thus 2/3 the length of the hex, plus any pad (in our case '==').

Implementation capabilities

The RSTP module is, without augmentation, capable of emulating STP as well as RSTP behavior. That RSTP behavior can be further augmented by MSTP, or both MSTP and SPB.

Given suitable YANG modules, a manager can discover a Bridge's spanning tree capabilities—which might, in theory at least, extend beyond the current possible set of rstp-mstp-spb (plus stp emulation).

My expectation is that the 'default' or unchanged state of a Bridge would be the sum of its capabilities, and force-protocol-version (if not explicitly managed) would reflect that. Per-port down selection reflects the capabilities of the Designated Port for the attached LAN (or the simple presence of an old STP Bridge).

The protocol version advertised by the Designated Port for the attached LAN is currently missing from rstp:bridge-port-parameters and should be added prior to rstp:bridge-port-parameters:root-id. It should have type uint8, not restricted to the enum subset specified for force-protocol-version, to accommodate any possible encoding in the received Protocol Version Identifier.

Raw data vs lexical representation

A Bridge ID, for example, is currently represented by the grouping:

```
grouping bridge-id {…
  container bridge-id {…
    leaf bridge-id {
      type uint64; …
    leaf bridge-priority {
      type id-priority; …
    leaf system-id-extension {
      type uint16 {
        range  "0..4095"; …
    leaf bridge-address {
      type ieee:mac-address; …
```

If I am running an app that fetches all the data, potentially from all the bridges in the network, runs computations and ‘what-ifs’, then all I need/want is the uint64 bridge-id leaf. The rest the app will work out for me.

If I am viewing the data, more or less as it is returned, then I want everything but the uint64, and I most certainly don’t want to see that in decimal.

Is having both views of that data an imposition on the managed system?