Support for Combo and Any PON

Goal

The goal of this document is to describe the requirements and design details for Combo and Any PON OLT support in VOLTHA and DMI.

1. Requirements

These are the high level requirements assumed for supporting Combo and Any PON OLT functionality in VOLTHA. The VOLTHA, in conjunction with DMI and the olt-agent running in the OLT, should work to adhere to these requi er naamrements.

1. The OLTs should come up with PON tech configuration as described in a configuration file, if one is available, else a default PON tech configuration for the given OLT vendor/model.

2. If the OLT supports dynamic configuration of PON tech, VOLTHA/DMI APIs should provide a mechanism for such dynamic configuration.
3. 4hIf the PON tech is dynamically reconfigured, OLT should record such configurations to be applied again on OLT reboots.
4. If the PON tech configuration cannot be applied on the OLT device, the agent/olt should clearly log/communicate the error/reason for such failure (eg: hw-failure, unsupported-config etc.) and the agent on the OLT should re-configure the PON to it previous working configuration.
5. It should be possible to query the PON tech, and logical-to-physical port mapping for Combo PON (if applicable) from VOLTHA/DMI NB APIs.
6. If the OLT offers APIs to understand the type of transceiver plugged into the PON port, those should be used and the type dynamically configured.

High Level Implementation Details

As per the aforementioned requirements for the Combo and Any PON feature, they translate to the following high level work items.

1. **Static configuration of PON tech.** The static configuration involves reading a PON tech configuration file on OLT startup by the agent running on the OLT, and then configuring the PON ports accordingly.
2. **Dynamic configuration** of PON tech. The dynamic configuration involves dynamically changing the PON port tech after the OLT is active.
3. **NB API changes** (voltha/dmi) to report PON tech, logical-to-physical port mapping for Combo PON (if applicable).

Supporting Static Configuration of PON Tech

PON Port configuration schematics

Below is the proposed port configuration schematics with an example. It defines two PON ports each having a different tech.

```
---
num_pon_ports: 2
ranges:
  -
    pon_id_range:
      start: 0
      end: 0
    tech: "XGS-PON"
    onu_id_range:
      start: 1
      end: 255
    alloc_id_range:
      start: 1024
      end: 8096
    gemport_id_range:
      start: 1024
```

```
    end: 8096
-
  pon_id_range:
    start: 1
    end: 1
  tech: "GPON"
  onu_id_range:
    start: 1
    end: 255
  alloc_id_range:
    start: 256
    end: 1024
  gemport_id_range:
    start: 256
    end: 1024
```

The `pon_id_range` can also be used to specify the same PON characteristics for a range of consecutive PON ports.

Changes to openolt-agent

The openolt-agent will be impacted with the following changes.

1. Reading the PON tech configuration file. The path of the configuration should default to some value and also be user configurable via command line arguments. For openolt-agent integrated with Broadcom BAL - use /broadcom path as default for PON port tech configuration file.
2. Validating the configuration file to see if the given OLT supports such a PON tech configuration schematics. If the configuration is invalid, it is treated as a FATAL error and the OLT shall not be active/operational and may require user intervention.
3. If the PON tech configuration file is valid, the agent shall invoke the OLT device APIs to configure the MAC and PON ports for the appropriate tech.
4. The agent should report the resource ranges (onu-id, alloc-id, gemport-id) per PON port when the openolt-adapter invokes the GetDeviceInfo API.

Given that openolt.proto interface message DeviceInfo already reports the resource ranges on a per PON port basis, no changes are needed in this aspect and hence no changes to any NB components in voltha in this aspect.

Changing PON tech for Combo PON OLTs that do not support dynamic config

If the PON tech is to be changed on the OLT, and the OLT does not support dynamic configuration, the following procedure describes the way to change the PON tech

1. The user load's the new PON tech configuration schematics file described in the aforementioned section at the path where the openolt binary looks for it. The file is loaded to the OLT out-of-band - meaning VOLTHA is not involved in this process.
2. Reboot the OLT

Once the OLT reboots the new PON tech scheme comes into effect. On OLT reboot - flows, queues, schedulers, pon resources are cleaned up.

Everything is re-setup once the OLT is reconnected per the new PON configuration.

Note: This process impacts services for the subscribers on all the PON ports of the OLT.

Dynamic Configuration of PON Tech

Dynamic configuration of PON tech allows changing PON tech for one or more PON ports on the OLT without impacting services on the remaining PON ports.
Device Management Interface (DMI) seems the right entry point for manipulating the PON tech to keep VOLTHA NB APIs agnostic to PON tech management.

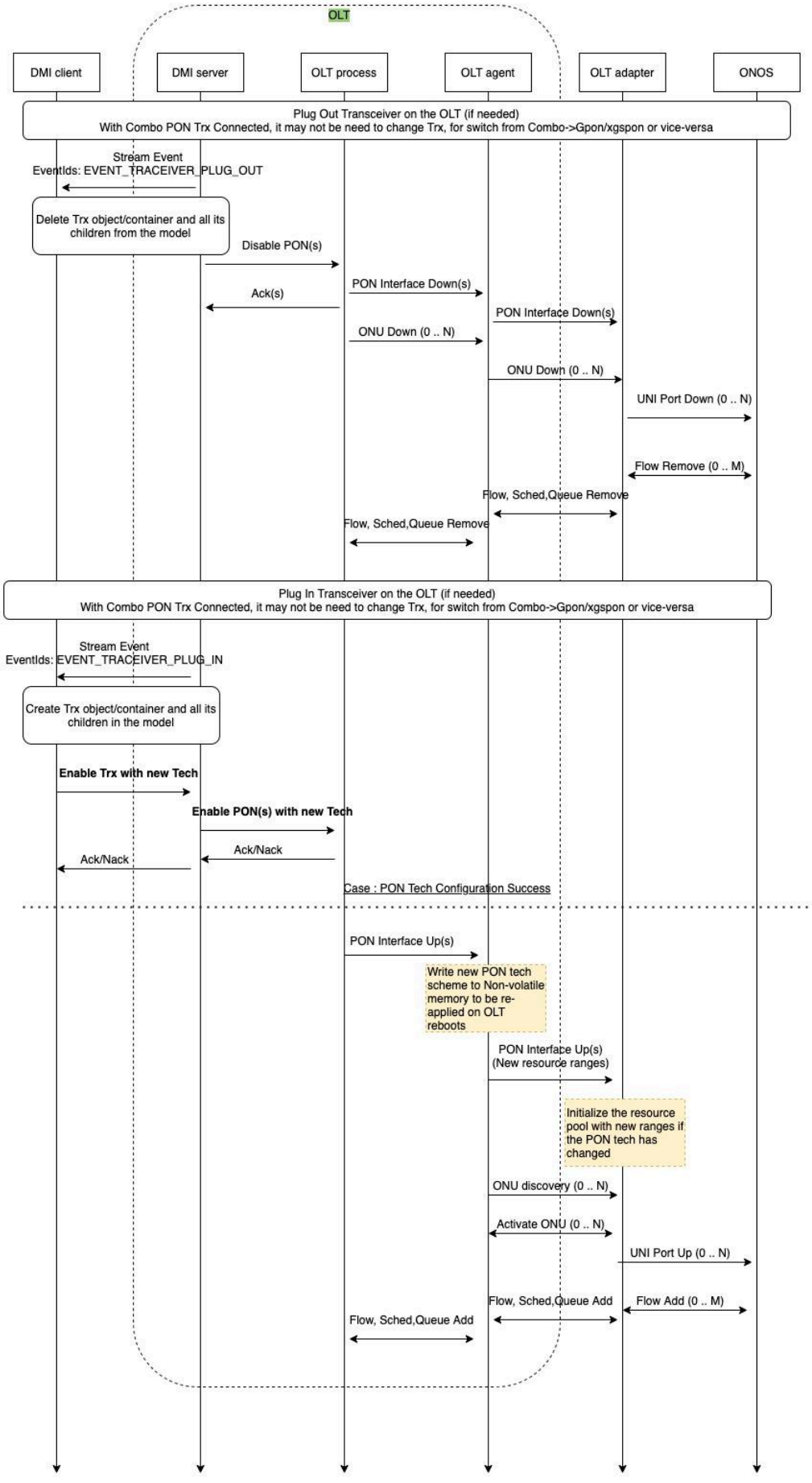
The dynamic configuration of PON tech involves the following steps from the DMI.

1. Disable the PON port for which the PON tech is to be modified.

2. Enable the PON port with the new PON tech setting.

Note: This is a service impacting procedure to all the ONUs on that PON port.

Below sequence diagram describes the dynamic configuration of PON tech.


OLT
DMI client
DMI server
OLT process
OLT agent
OLT adapter
ONOS
Plug Out Transceiver on the OLT (if needed)
With Combo PON Trx Connected, it may not be need to change Trx, for switch from Combo->Gpon/xgspon or vice-versa
Stream Event
EventIds: EVENT_TRACEIVER_PLUG_OUT
Delete Trx object/container and all its children from the model
Disable PON(s)
PON Interface Down(s)
Ack(s)
PON Interface Down(s)
ONU Down (0 .. N)
ONU Down (0 .. N)
UNI Port Down (0 .. N)
Flow Remove (0 .. M)
Flow, Sched,Queue Remove
Flow, Sched,Queue Remove
Plug In Transceiver on the OLT (if needed)
With Combo PON Trx Connected, it may not be need to change Trx, for switch from Combo->Gpon/xgspon or vice-versa
Stream Event
EventIds: EVENT_TRACEIVER_PLUG_IN
Create Trx object/container and all its children in the model
Enable Trx with new Tech
Enable PON(s) with new Tech
Ack/Nack
Ack/Nack
Case : PON Tech Configuration Success
PON Interface Up(s)
Write new PON tech scheme to Non-volatile memory to be re-applied on OLT reboots
PON Interface Up(s)
(New resource ranges)
Initialize the resource pool with new ranges if the PON tech has changed
ONU discovery (0 .. N)
Activate ONU (0 .. N)
UNI Port Up (0 .. N)
Flow, Sched,Queue Add
Flow Add (0 .. M)
Flow, Sched,Queue Add

Below describes the more details of the sequence diagram

Step 1:

Plug out the trx. This generates an `Event` (check `rpc StreamEvent` in `hw_events_mgmt_service.proto`) with EventIds as `EVENT_TRANSCEIVER_PLUG_OUT`.
The Trx container and all its port objects are cleared from the model.
The DMI server then triggers a PON port disable for the PON ports mapped to the transceiver that was plugged out.

Step 2:

When the PON port is down, the olt-agent indicates the olt adapter about PON Interface down via `IntfOperIndication` message.

Check openolt.proto for details.This is followed by `OnuIndication` (down) message for ONUs on that PON port. Once the ONU is down, the uni port is marked as Down (or disabled) and this removes all flows, queues and schedulers.

Step 3:

Plug in the new trx. This generates an `Event` (check `rpc StreamEvent` in `hw_events_mgmt_service.proto`) with EventIds as `EVENT_TRANSCEIVER_PLUG_IN`. The Trx container and all its port objects are created in the model.

Step 4:

The DMI client triggers the `SetHWComponentInfo` API for enabling the transceiver with the new tech. This is not needed if the technology can be automatically detected from the transceiver and configured to the transceiver object. If the technology is

automatically detected, then this step can be skipped.

```
// The attributes of a component which are modifiable from the client side
message ModifiableComponent {
     // The name has to be unique for each component within hardware and implementations need to
     // ascertain this when modifying the name
    string name = 1;
    ComponentType class = 2;
    Component parent = 3;
    int32 parent_rel_pos = 4;
    string alias = 5;
    string asset_id = 6;
    Uri uri = 7;
    ComponentAdminState admin_state = 8;
      // The attribute 'specific' can be populated for specific class of components
    oneof specific {
                PortComponentChangeAttributes port_attr = 50;

TransceiverComponentChangeAttributes trx_attr = 51;
```

```
    }
}

enum TransceiverType {
    TYPE_UNDEFINED = 0;
    ETHERNET = 1;
    GPON = 2;
    XGPON = 3;
    XGSPON = 4;
    CPON = 5;
    NG_PON2 = 6;
    EPON = 7;
    // Add more here

    TYPE_NOT_DETECTED = 255;
}


message
TransceiverComponentChangeAttributes {
    TransceiverType trans_type = 1;
}
```

Note: Only important/relevant attributes are documented.

`HWComponentInfoSetRequest.ModifiableComponent.ComponentType = COMPONENT_TYPE_TRANSCEIVER`

`HWComponentInfoSetRequest.ModifiableComponent.TransceiverComponentChangeAttributes.TransceiverType = <new-pon-tech>`

Step 5:

The status of the Trx Tech configuration is indicated in the `HWComponentInfoSetResponse` response message of the `SetHWComponentInfo` API.

If the result is a failure, the DMI server has to handle the error code. If the error code cannot be gracefully handled this may need manual intervention.

If the result is a success, The component PON ports of the trx are configured with the new

technologies. The PON `IntfOperIndication` (Up) is indicated to olt-adapter with the new PON tech and resource ranges. The highlighted attributes are to be newly added to support this functionality.

```
message IntfOperIndication {
    string type = 1;        // nni, pon
    fixed32 intf_id = 2;
    string oper_state = 3;       // up, down
    fixed32 speed = 4;       // measured in
Mbps


    bool pon_tech_updated = 5; // Relevant
when oper_status is "up" and port "type" is
"pon".
                                // Set to
true if PON tech is updated.

    string technology = 6;

    message PONResourceRanges {

        message Pool {

            enum PoolType {
                ONU_ID = 0;
                ALLOC_ID = 1;
```

```
                GEMPORT_ID = 2;
                FLOW_ID = 3;
            }


            PoolType type = 1;
        fixed32 start = 3; // lower bound
on IDs allocated from this pool
        fixed32 end = 4; // upper bound on
IDs allocated from this pool
    }

        repeated Pool pools = 3;
    }

    PONResourceRanges ranges = 7;

}
```

The olt-agent should also write the new PON tech config to non-volatile-memory so that this new tech config is applied to PON ports on OLT reboots instead of the factory setting PON tech config.

The adapter should initialize the resource pools with the new ranges.

Exposing Combo PON information via DMI NB APIs

The PON information can be fetched from the `HWComponentInfoGetRequest` API on the PON Transceiver object.

```
message Component {

    // The name of a component uniquely
identifies a component within the hardware

    string name = 1;
    ComponentType class = 2;
    string description = 3;
    // The name of the parent of this
component, empty string("") in case of the
root component
    string parent = 4;
    int32 parent_rel_pos = 5;
    repeated Component children = 6;
    string hardware_rev = 7;
    string firmware_rev = 8;
    string software_rev = 9;
    string serial_num = 10;
    string mfg_name = 11;
    // Apart from the definition of this
attribute as defined in RFC 8348,
implementations could choose to carry
```

```
    // the manufacturer's part number in
this attribute.
    string model_name = 12;
    string alias = 13;
    string asset_id = 14;
    bool is_fru = 15;
    google.protobuf.Timestamp mfg_date =
16;
    Uri uri = 17;
    // The uuid of the component uniquely
identifies the component across the entire
system
    Uuid uuid= 18;
    ComponentState state = 19;
    repeated ComponentSensorData
sensor_data = 20;
    // The attribute 'specific' can be
populated for components where more details
are required by the users of the DMI
interface

    oneof specific {
        PortComponentAttributes port_attr =
50;
        ContainerComponentAttributes
container_attr = 51;
        PsuComponentAttributes psu_attr =
52;
        TransceiverComponentsAttributes
transceiver_attr = 53;
    }
```

```
}
```

Note1: There could be more than one `Component` of `ComponentType` `COMPONENT_TYPE_PORT` being a child of `Component` of `ComponentType` `COMPONENT_TYPE_TRANSCEIVER` in which case, the Port `Component` is part of Combo PON set.

Note2: It is important to note that the `mapping_label` attribute in `PortComponentAttributes` message gives the mapping of the DMI port object to Voltha `Port` object.

Phased approach for feature development

The Combo PON development using real hardware is generally going to take more time given that it needs timely support from OLT device

vendors (both on the hardware and software aspects) and also procuring the Combo PON transceivers. So doing the initial development and writing voltha-system-tests with BBSIM as a stub for a real OLT could help develop and test other components in the VOLTHA/DMI system much faster.

Additional Notes

High Level Details on how Broadcom Aspen 65650 MAC provides Combo PON functionality

The way Combo PON tech is realized in Broadcom Aspen 68656/68658 MAC is two different PON technologies, say GPON and XGS PON, is configured on two different PON port from the BAL API on the MAC device and then the data from these two PON ports are channeled on a single external facing PON port of the OLT. For

example, PON0 is configured with XGS PON and PON1 is configured with GPON and then these two PON ports are paired internally (on the OLT MAC) and traffic is routed through PON0 that faces outside on the OLT. The external facing PON0 should be equipped with a Combo PON transceiver. Now this PON0 will host both XGS PON and GPON ONUs. Internally, the traffic is divided and processed as two different PON ports on the MAC, but externally it will be visible as a single Optical Distribution Network (ODN) through a single PON. Note that the paired PON ports need to be on the same MAC device in the OLT.

Decoding EEPROM data of the transceiver

The EEPROM data has details about the SFP+ or QSFP+ transceiver that could help in detecting the following important information (among other things)

1. Vendor name

2. Vendor OUI
3. Vendor Part Number
4. Vendor revision
5. Downstream Wavelength

Some or all of the information above could be used to automatically detect the technology of the plugged transceiver.
Generally most of the SFP+ or QSPF+ are compliant to the following specifications from SNIA which describes details about decoding the EEPROM data

1. **SFF-8472**: Relevant for SFP+
2. **SFF-8436**: Relevant for QSFP+

On whiteboxes based on ONL, the ONLP utility can be used to dump the EEPROM data.

Examples

Example 1:

Below is the EEPROM dump of the SFP+ connected on the RadiSys 3200G OLT. **SFF-8472** specification

is used to decode some important details about the SFP+

```
eeprom:
  0000: 03 04 01 00 0a 00 00 00 00 00
00 03 19 00 14 c8
  0010: 00 00 00 00 48 69 73 65 6e 73
65 20 20 20 20 20
  0020: 20 20 20 20 00 ac 4a fe 4c 54
45 33 36 38 30 4d
  0030: 2d 42 43 2b 20 20 20 20 31 31
20 20 05 d2 00 c6
  0040: 00 1c 14 14 55 39 54 41 31 30
30 30 30 33 39 20
  0050: 20 20 20 20 32 30 30 31 31 36
20 20 58 e0 01 07
  0060: 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
  0070: 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
  0080: ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff
```

```
  0090: ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff
  00a0: ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff
  00b0: ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff
  00c0: ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff
  00d0: ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff
  00e0: ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff
  00f0: ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff
```

bytes	data	key	value (decoded)
20-35	48 69 73 65 6e 73 65 20 20 20 20 20 20 20 20 20	Vendor name (in ASCII)	Hisense

37-39	ac 4a fe	Vendor OUI	Ac:4a:fe is the OUI of Hisense
40-55	4c 54 45 33 36 38 30 50 2d 42 43 2b 20 20 20 20	Vendor P/N (in ASCII)	LTE3680P-BC+
56-59	31 31 20 20	Vendor Rev (in ASCII)	11
60-61	05 d2	Wavelength	1490nm (**GPON**)

Example 2:

Below is the EEPROM dump of the QSFP+ connected to the Edgecore ASxVOLT16 OLT. **SFF-8436** specification is used to decode some important details about the QSFP+

`eeprom:`

```
  0000: 06 20 55 00 f1 00 4e 00 f8 00
8d cc 74 04 87 8c
  0010: 7a 44 88 b8 00 00 7e f4 00 fa
7b 3c 13 88 61 e3
  0020: 18 97 0f 8d 00 09 0c 5a 00 0b
00 00 00 00 00 00
  0030: 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
  0040: 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
  0050: 00 00 00 00 a2 00 00 00 00 c0
00 c0 1c 00 0f 0f
  0060: 15 23 83 99 59 dd 37 da 02 0d
00 00 00 00 20 a0
  0070: 00 00 00 00 00 00 00 ff ff ff
ff ff ff ff ff 01
  0080: 06 b0 01 00 00 00 00 00 00 00
00 90 63 63 14 00
  0090: 00 00 00 f6 4c 69 67 65 6e 74
20 50 68 6f 74 6f
  00a0: 6e 69 63 73 80 00 00 00 4c 54
48 37 32 32 36 2d
```

```
  00b0: 50 43 20 20 20 20 20 20 30 31
7b 34 03 e8 4b 50
  00c0: 96 ff 8a 00 48 32 37 37 32 5a
30 30 30 32 30 20
  00d0: 20 20 20 20 31 37 30 32 31 35
20 20 08 40 00 dd
  00e0: 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
  00f0: 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
```

bytes	data	key	value (decoded)
148-163	4c 69 67 65 6e 74 20 50 68 6f 74 6f 6e 69 63 73	Vendor name (in ASCII)	Ligent Photonics
165-167	00 00 00	Vendor OUI	00:00:00 is the OUI of Ligent (seems

			some default and not filled properly by the manufacturer)
168-183	4c 54 48 37 32 32 36 2d 50 43 20 20 20 20 20 20	Vendor P/N (in ASCII)	LTH7226-PC
184-185	30 31	Vendor Rev (in ASCII)	01
186-187	7b 34	Wavelength (in 0.05 nm resolution)	1577 nm (**XGS-PON**)