BUILDING A TRANSPARENT LAYER 1 SWITCH USING P4

A lightning talk about an internal research „project“

Sebastian Graf | DENOG12 | 10.11.2020

Xantaro SERVICE INTEGRATION

DISCLAIMER

This talk is not an introduction to the P4 language. You will not see any P4 code. Consider the following resources for that:

- Powerful Properties of Packet Processing with P4 - Aaron A. Glenn – DENOG10
  - https://github.com/denog/media/tree/master/DENOG10
  - https://media.ccc.de/c/denog10
- p4.org
- github.com/p4lang/

PROBLEM STATEMENT

- We operate a multi-vendor lab network
  - Automation for Network Lab Environments – Tobias Heister – DENOG9
    - https://github.com/denog/media/tree/master/DENOG9
- A lot of pre-cabled standard topologies for different scenarios, but every now and then somebody comes up with a cabling requests were we run into issues
  - Out-of-ports error
  - Out-of-transceiver error
  - Out-of-time error
  - Out-of-budget error
- Obviously, we also need somebody on site to actually do the change

EASY SOLUTION

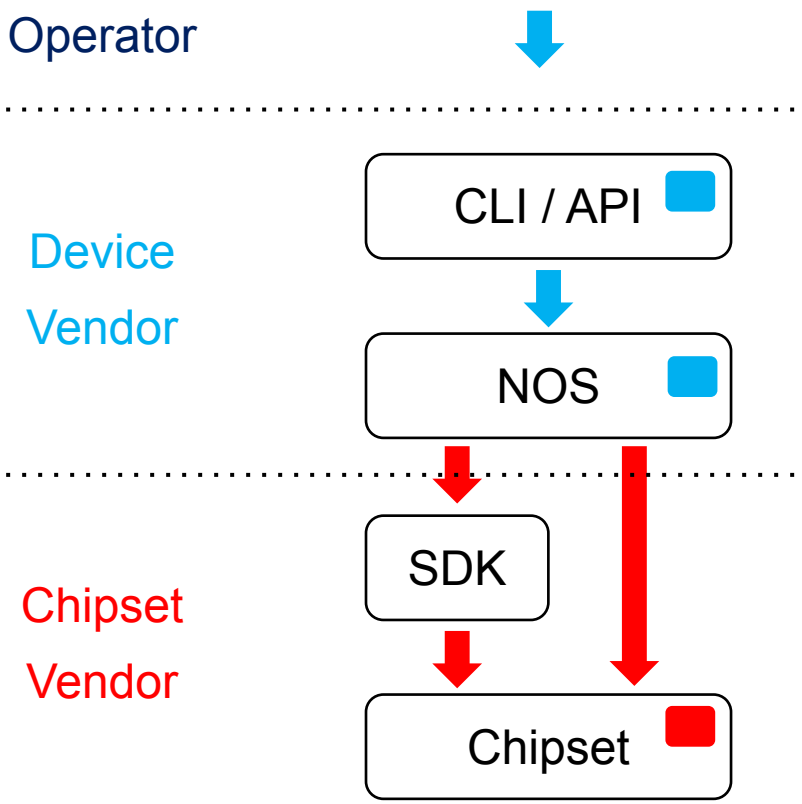
- So what do you do, when you have not enough ports on your expensive devices?
  - Put a switch between them
- In our case we grabbed some Juniper QFX Switches originally put into the Lab for a Qfabric deployment
  - Started as Virtual Chassis with QFX3500 and QFX3600
  - Added EX4300 later on for 1G support
  - Added two variants of QFX5100 later on for additional 10G and 40G Ports
  - Allows user-definable connections between lab devices by configuring a vlan
  - And yes, we actually successfully performed multiple ISSUs on our Frankenfabric!
- But there are issues with that
  - Quite ok for Layer 3 and above, but not really transparent for Layer 2
  - You always find an ugly protocol, that gets eaten by the switch
  - It is a switch, so it behaves like one (mac learning, intercepts frames to control plane that it considers interesting)

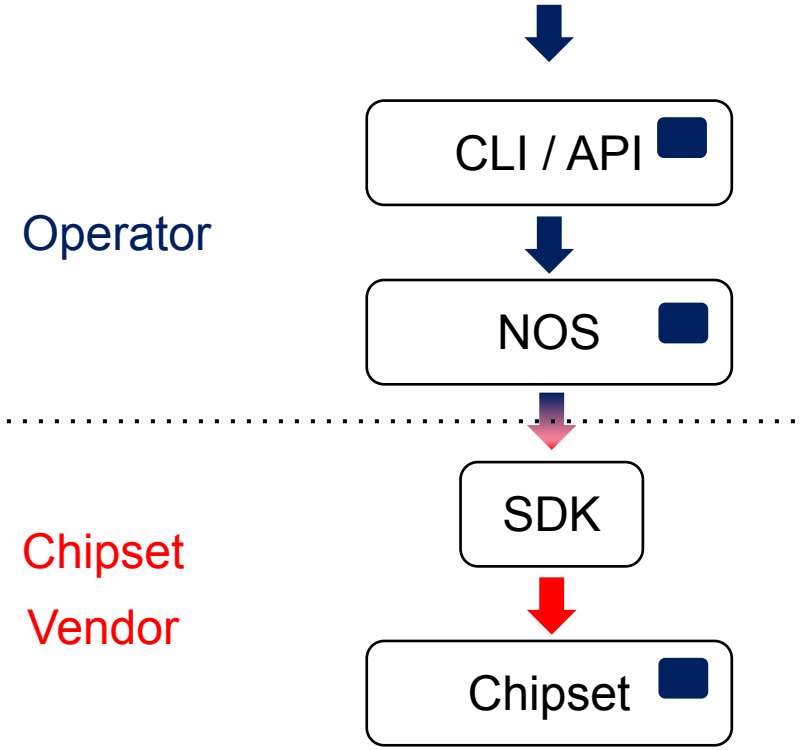
LET‘S GO SHOPPING

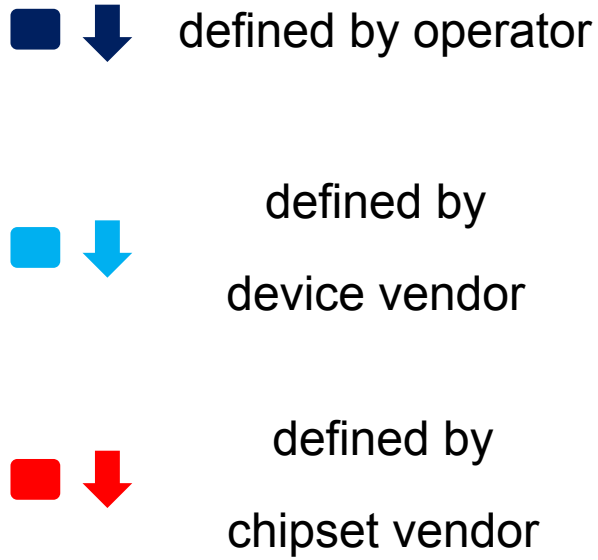
- There are ready-to-use solutions for Layer 1 switching on the market
- Typically employ one of the following principles
  - Patch robots
    - Takes quite some rackspace
    - Need to decide on physical media (singlemode or multimode, LC / MTP)
    - As transparent as a cable can be
  - Optical-Electrical-Optical
    - Can transform between media types
    - Operate on bit layer and hence provide transparency
- After getting quotes to have an interconnect matrix for some 100G Ports, we looked for alternatives
  - We had two P4-capable WEDGE 100BF-32X switches in the lab, originally purchased for whitebox testing

P4 IN ONE SLIDE

- Compared to a “classical” network device, a P4 enabled one gives you a lot of opportunities but also work
- In the classical world, somebody else tells you how to instruct the device to do stuff







HIGH LEVEL STEPS TO GET P4 RUNNING ON THE WEDGE 100BF-32X

- When you power-up a Wedge 100BF-32X it defaults to be a doorstopper with 32 QSFP28 Ports
- You need to program it with P4. Writing a P4 program that forwards frames based on incoming port is quite simple, but in order to get this up and running you need to follow some steps
  - The switch boots into Open Network Install Environment (ONIE), you need to load a NOS on it
  - We decided to use Open Network Linux (ONL), so we had to
    - Clone ONL repo, apply patch from chipset vendor
    - Build the image
    - Upload the image to the switch and use ONIE to install it
  - Copy hardware specific packages to the switch (running ONL) and install them
  - Compile / install the SDK from the chipset vendor and install it on ONL
  - Write P4 program and tell the SDK to execute it
  - Change port speed (10G breakout is default) and enable them
  - Insert entries into the lookup table to create forwarding entries

CHECK TRANSPARENCY

- Once a packet reaches the chipset, it is stored as series of 0s and 1s
  - You have to write a parser definition for everything that it should process
  - As the chipset does not care about any headers without a corresponding parser, we suspected that it should be quite transparent if we do not tell it how to process any protocol headers
- First transparency tests (with varying legality of frames) done with packet generator
  - LACP went through without an issue, other ugly stuff as well
  - Ethernet frames using ff:ff:ff:ff:ff:ff as source MAC were forwarded
  - Frames with totally messed up ethernet headers (but correct crc) were forwarded
  - Sending frames with incorrect CRC got dropped => never reached the chipset for processing
  - Linerate with 99,9995% traffic rate, latency: Average: 768ns, Minimum: 737 ns, Max: 812ns
    - Latency roughly equals to 150 meters of singlemode fiber
  - Second test when a colleague requested a cable between two Juniper MX during a PoC
  - Used the P4 program as replacement for a direct connection

CONCLUSION

- We used a quite powerful device for a quite boring task
  - But it works out nicely and is quite compelling from a price perspective
  - Now gathering ideas what we could add as features
    - Traffic amplification
    - Local and remote mirroring
    - Statistical packet drops
    - ...
  - P4 would even allow us to define our own dataplane encapsulations if we had to
    - Still searching for a reason to do that....