SONiC Software Subsystems Overview

OCP workshop Aug 2018

Rodny Molina

Linkedin

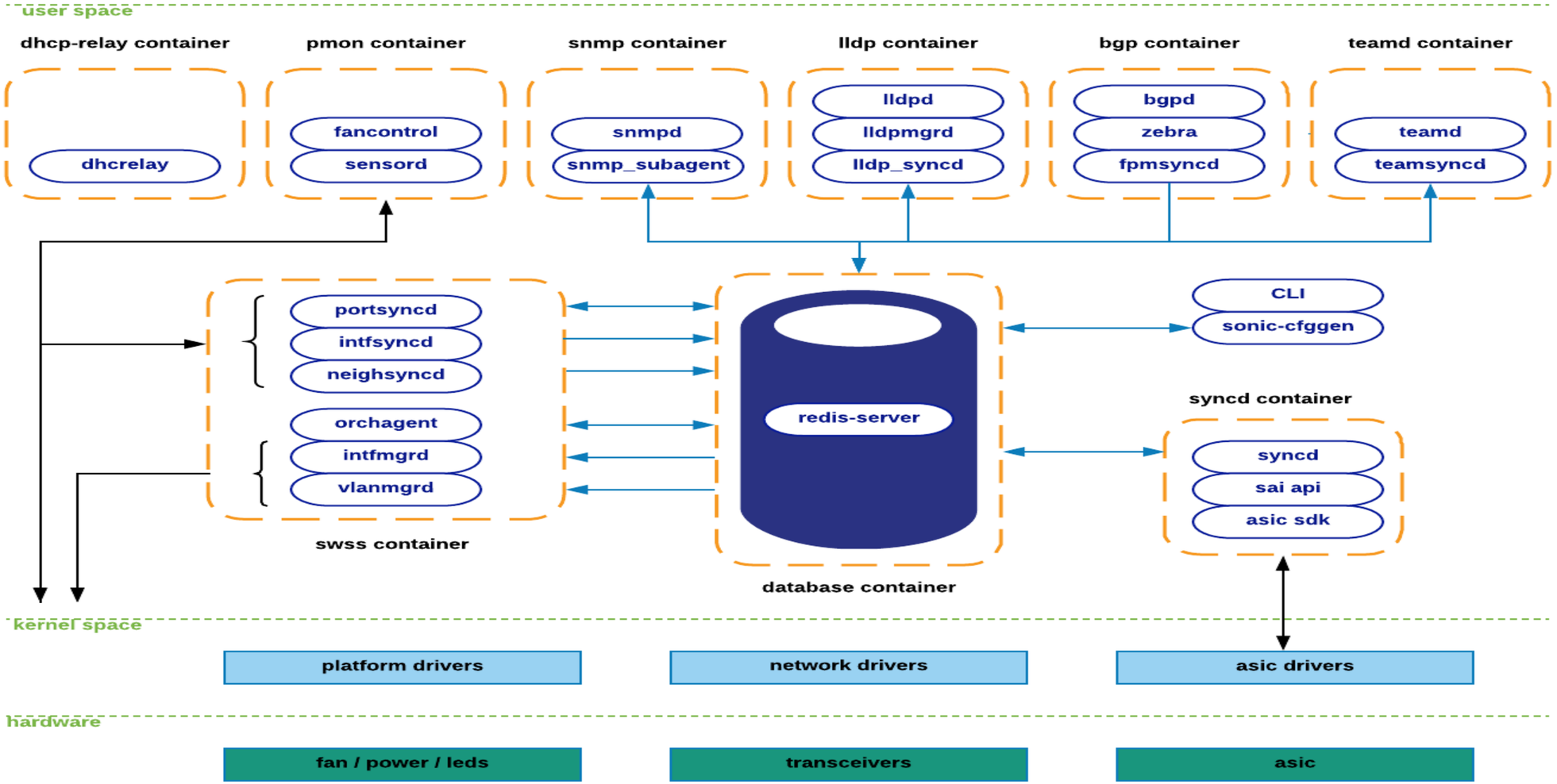
rmolina@linkedin.com

agenda

- SONiC Software Subsystems.
- SONiC Subsystems Interactions.
- SONiC Modularity -- FRR example.
- SONiC CLI Overview.

SONiC Software Subsystems

Software Subsystems High-level View



Platform Monitor container (pmon)

- sensord: Daemon used to periodically log sensor readings from hardware components and to alert when an alarm is signaled.
- fancontrol: Process to collect temperature stats from platform sensors and react on that by increasing/decreasing fan’s spinning speed.

SNMP container

- snmpd: Actual snmp server (master) in charge of handling incoming snmp polls from external network elements.
- snmp-agent (sonic_ax_impl): This is SONiC's implementation of an AgentX snmp subagent. This subagent feeds the master-agent (snmpd) with information collected from SONiC databases in the centralized redis-engine.
- snmp-agent subscribes to the following databases/tables:
  - APPL_DB: PORT_TABLE, LAG_TABLE, LAG_MEMBER_TABLE, LLDP_ENTRY_TABLE
  - STATE_DB
  - COUNTERS_DB
  - ASIC_DB: ASIC_STATE:SAI_OBJECT_TYPE_FDB

LLDP container

- Lldp: Actual lldp daemon featuring lldp functionality. This is the process establishing lldp connections with external peers to advertise/receive system capabilities.
- Lldp_syncd: Process in charge of uploading lldp’s discovered state to the centralized system’s message infrastructure (redis-engine). By doing so, lldp state will be delivered to applications interested in consuming this information (e.g. snmp).
- Lldpmgr: Process provides incremental-configuration capabilities to lldp daemon; It does so by subscribing to CONFIG_DB and STATE_DB within the redis-engine.

Routing container (bgp)

- bgpd: regular bgp implementation. Routing state from external parties is received through regular tcp/udp sockets, and pushed down to the forwarding-plane through the zebra/fpmsyncd interface.
- zebra: acts as a traditional IP routing-manager by providing routing-table updates, interface-lookups and route-redistribution services across different protocols. Zebra also takes care of pushing the computed FIB down to both kernel (through netlink interface) and to south-bound components involved in the forwarding process (through Forwarding-Plane-Manager interface –FPM--).
- fpmsyncd: small daemon in charge of collecting the FIB state generated by zebra and dumping its content into the Application-DB table (APPL_DB) seating within the redis-engine.

Switch State Services container (swss) (1)

- portsyncd: Listens to port-related netlink events and transfer information to APPL_DB. Attributes such as port-speeds, lanes and MTU are transferred through this channel. Portsyncd also inject state into STATE_DB.
- Intfsyncd: Listens to interface-related netlink events and push collected state into APPL_DB. Attributes such as new/changed ip-addresses associated to an interface are handled by this process.
- neighsyncd: Listens to neighbor-related netlink events triggered by newly discovered neighbors as a result of ARP processing. This state will be eventually used to build the adjacency-table required in the data-plane for L2-rewrite purposes.

Switch State Services container (swss) (2)

- orchagent: The most critical component in the SwSS subsystem. Orchagent contains logic to extract all the relevant state injected by *syncd daemons, process and massage this information accordingly, and push it through its south-bound interface.
- intfMgrd: Reacts to state arriving from APPL_DB, CONFIG_DB and STATE_DB to configure interfaces in the linux kernel.
- vlanMgrd: Reacts to state arriving from APPL_DB, CONFIG_DB and STATE_DB to configure vlan-interfaces in the linux kernel.

Syncd container

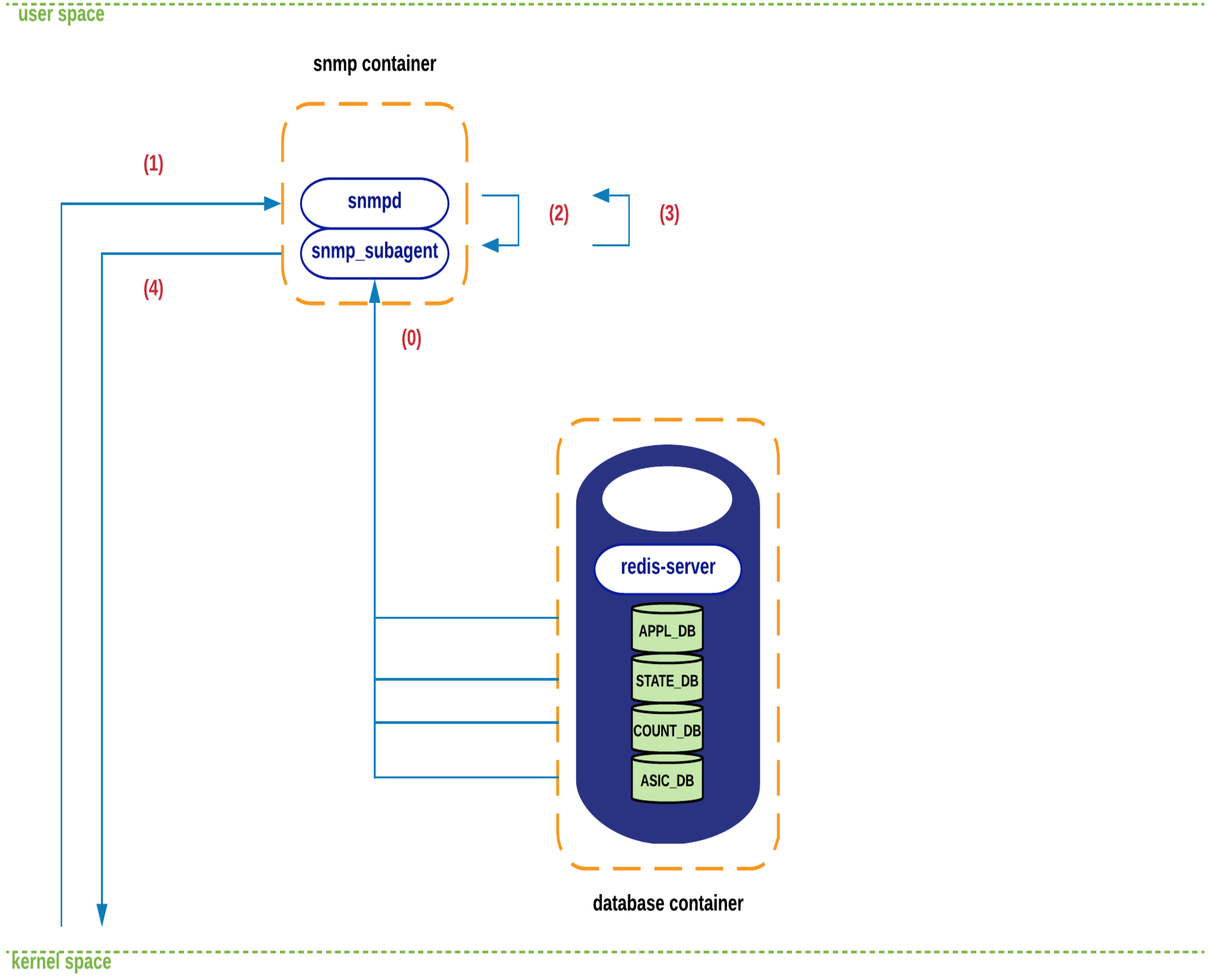
- syncd’s container goal is to provide a mechanism to allow the synchronization of the switch’s network state with the switch’s actual hardware/ASIC. This includes the initialization, the configuration and the collection of the switch’s ASIC current status.
- Main logical components:
  - syncd: Process in charge of executing the synchronization logic mentioned above. At compilation time, syncd links with the ASIC SDK library provided by the hardware-vendor, and injects state to the ASICs by invoking the interfaces provided for such effect.
  - SAI API: The Switch Abstraction Interface (SAI) defines the API to provide a vendor-independent way of controlling forwarding elements, such as a switching ASIC, an NPU or a software switch in a uniform manner.
  - ASIC SDK: Hardware vendors are expected to provide a SAI-friendly implementation of the SDK required to drive their ASICs. This implementation is typically provided in the form of a dynamic-linked-library which hooks up to a driving process (syncd in this case) responsible of driving its execution.

Database container

- Hosts the redis-database engine. Databases held within this engine are accessible to SONiC applications through a UNIX socket.
- APPL_DB: Stores the state generated by all state producers – routes, next-hops, neighbors, interfaces, etc. This is the south-bound entry point for all applications wishing to interact with other SONiC subsystems.
- CONFIG_DB: Stores the configuration state created by SONiC applications – port configurations, interfaces, vlans, etc.
- STATE_DB: Stores “key” operational state for entities configured in the system. This state is used to resolve dependencies between different SONiC subsystems. Basically, this DB stores all the state that is deemed necessary to resolve cross-modular dependencies.
- ASIC_DB: Stores the necessary state to drive asic’s configuration and operation – state here is kept in an asic-friendly format to ease the interaction between syncd (see details further below) and asic SDKs.
- COUNTERS_DB: Stores counters/statistics associated to each port in the system. This state can be utilized to satisfy a CLI local request, or to feed a telemetry channel for remote consumption.

SONiC Module Interactions

SNMP state interactions



(0) Initialization of MIB subcomponents supported in snmp-subagent process, and fetching of all state into local cache. Information is refreshed every few seconds to ensure that DBs and snmp-subagent are in-sync.

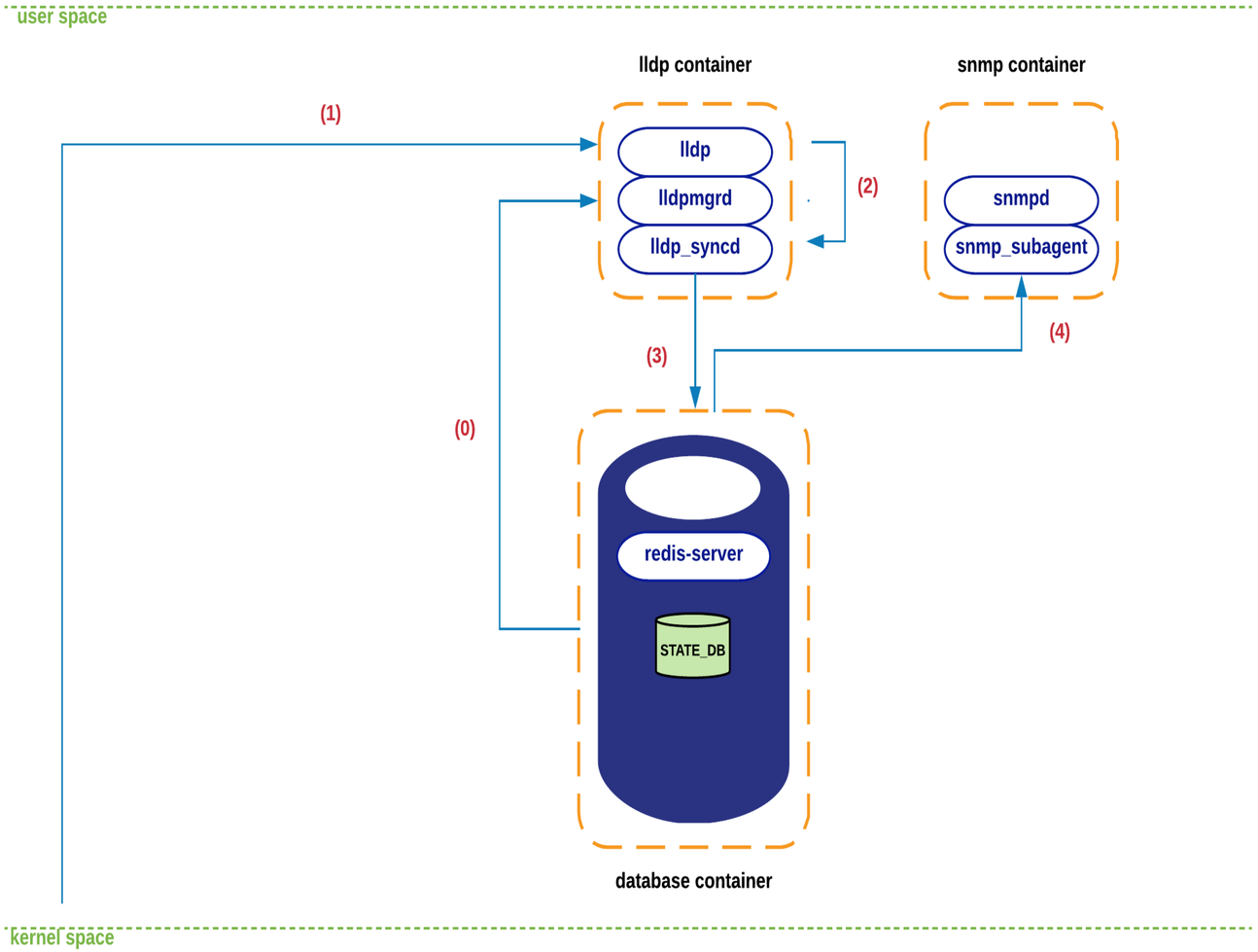
(1) A snmp query arrives at snmp's socket in kernel space. Kernel's network-stack delivers the packet to snmpd process.

(2) The snmp message is parsed and an associated request is sent towards SONiC's agentX subagent.

(3) Snmp-subagent serves the query out of the state cached in its local data-structures, and sends the information back to snmpd process.

(4) Snmpd eventually sends a reply back to the originator through the usual socket interface.

LLDP state interactions



(0) During initialization lldpmgrd subscribes to STATE_DB to get a life-feed of the state of the physical ports in the system. Based on this information, lldpd (and its network peers), will be kept aware of changes in the system's port-state and any configuration change affecting its operation.

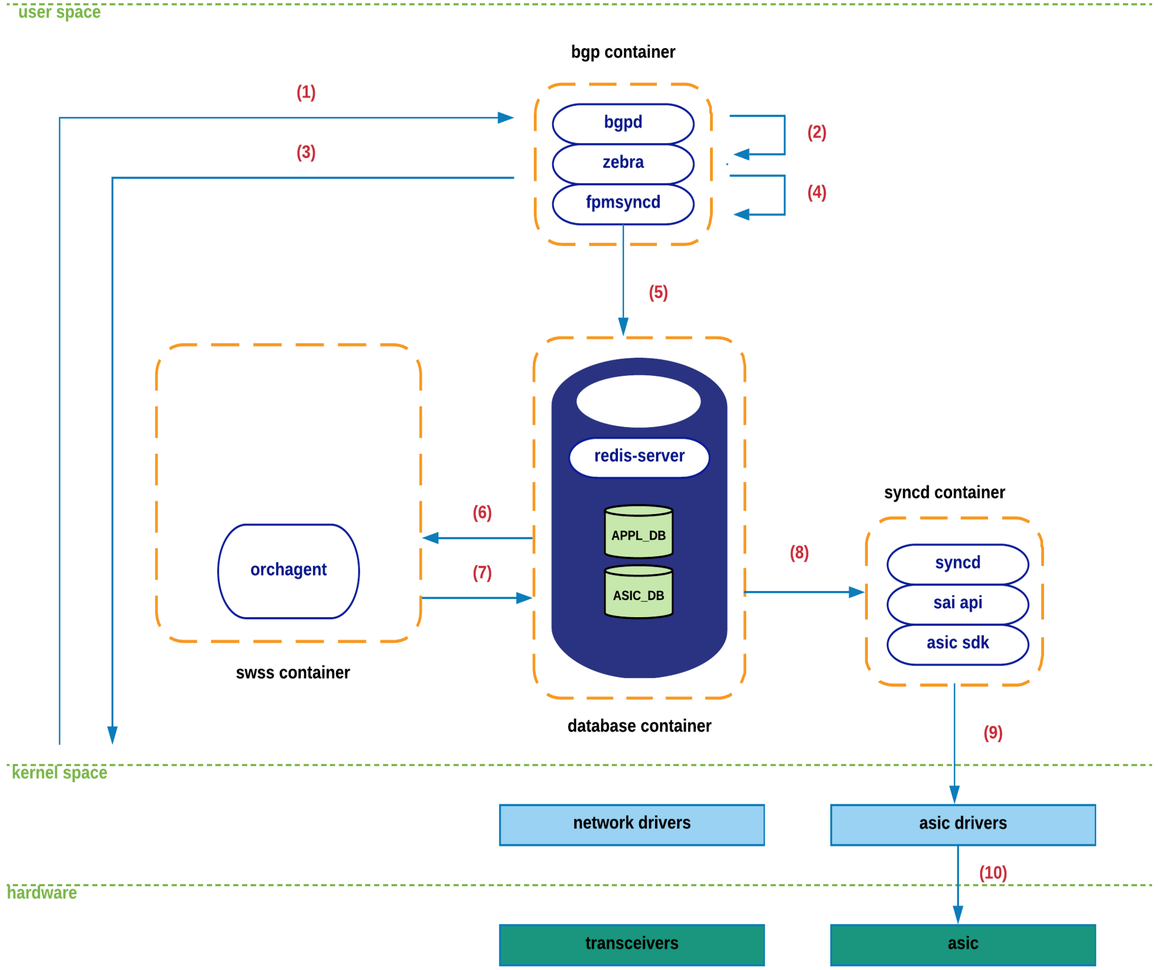
(1) A new LLDP packet arrives at lldp's socket in kernel space and this one delivers its payload to lldp process.

(2) Lldp parses this new state, which is eventually picked up by lldp_syncd during its execution of lldpctl cli command -- which typically runs every 10 seconds.

(3) Lldp_syncd pushes this new state into APPL_DB, concretely to table LLDP_ENTRY_TABLE.

(4) From this moment on, all entities subscribed to this table should receive a copy of the new state (currently, snmp is the only interested listener).

Routing state interactions



(1) A bgp-update arrives at bgp's socket in kernel and eventually to bgpd process.

(2) bgpd processes the msg and notifies zebra of the existence of a new prefix and associated next-hop.

(3) Upon determination by zebra of the feasibility/reachability of this prefix, it generates a route-netlink message to inject new state in kernel.

(4) zebra makes use of the FPM interface to deliver this netlink-route message to fpmsyncd.

(5) fpmsyncd pushes this state into APPL_DB.

(6) orchagentd receives the content of the information previously pushed to APPL_DB.

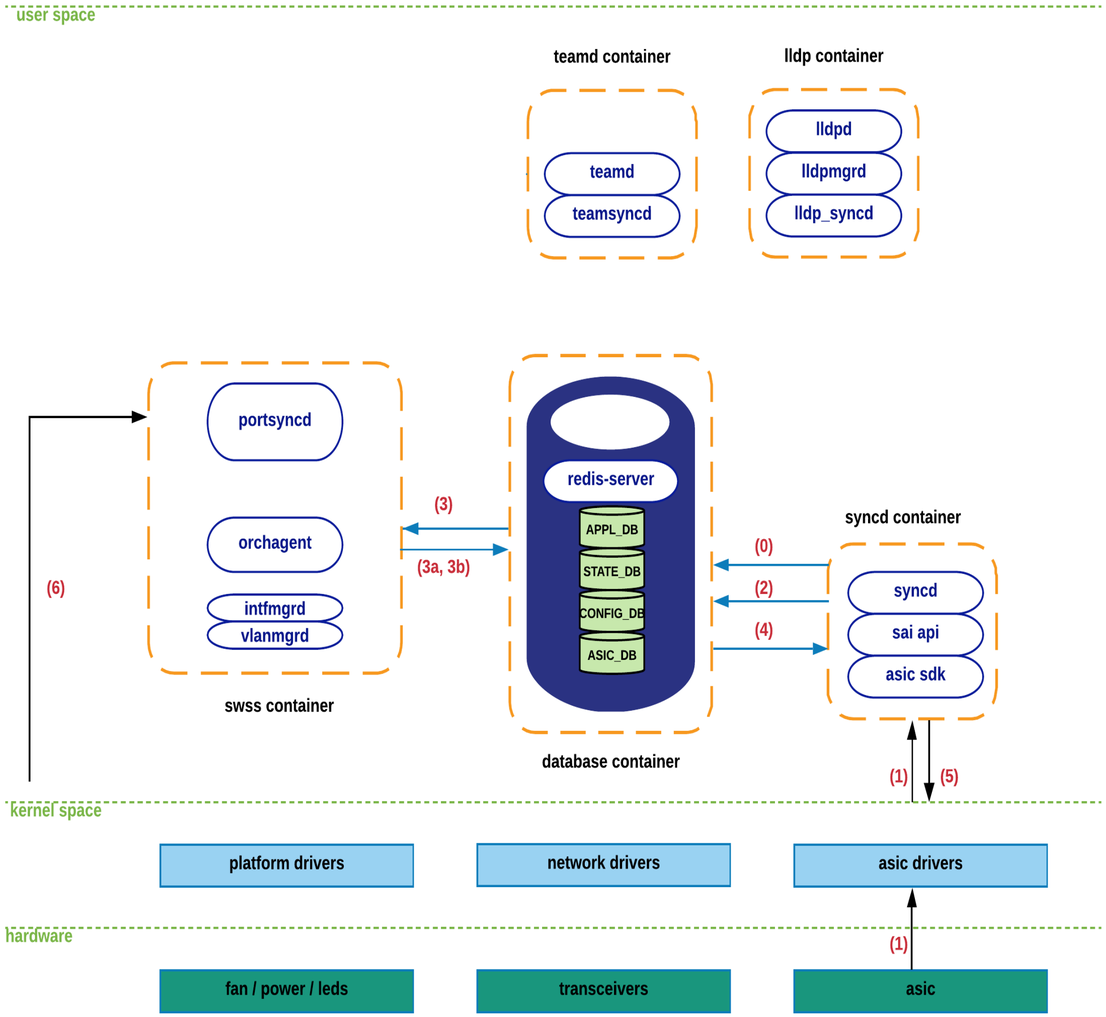
(7) orchagent processes the received information and invoke sairedis APIs to inject the new state into ASIC_DB.

(8) syncd receives the new state generated by orchagentd.

(9) syncd invoke SAI APIs to inject this state into the corresponding asic-driver.

(10) New route is finally pushed to hardware.

Port state interactions (1)



(1) Loss-of-carrier by ASIC's optical module. Notification is sent to the associated driver and syncd.

(2) syncd invokes the proper notification-handler and sends the port-down event towards ASIC_DB.

(3) orchagent makes use of its notification-handler to collect the new state from ASIC_DB, and executes the port-state-change sequence to:

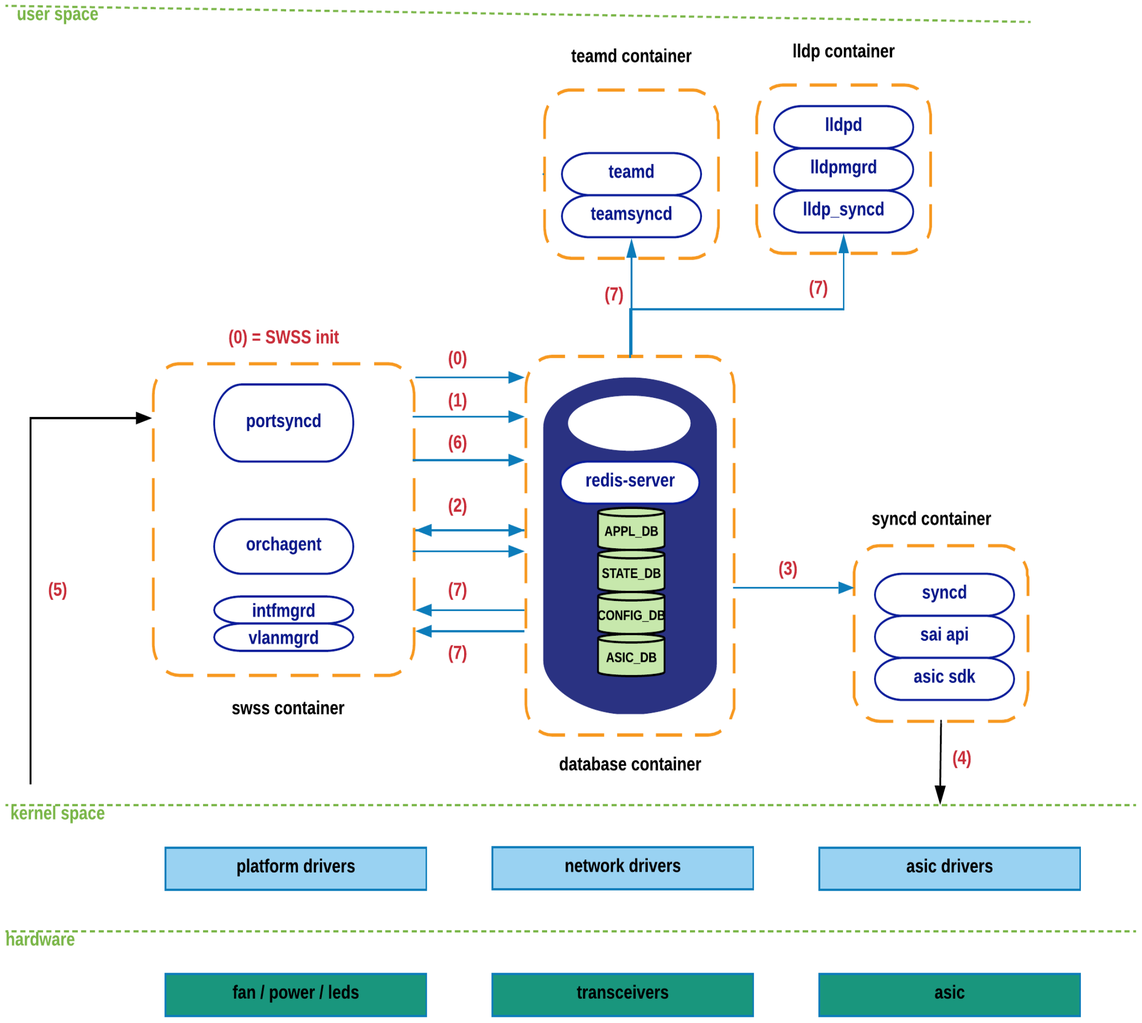
(3.a) Generate an update to APPL_DB to alert applications relying on this state for their operation.
(3.b) Invoke sairedis APIs to alert syncd of the need to update the kernel state associated to the host-interface of the port being brought down.

(4) syncd receives this new request through ASIC_DB and prepares to invoke the SAI APIs required to satisfy orchagent's request.

(5) syncd uses SAI APIs + ASIC SDK to update the kernel with the latest operational state (DOWN) of the affected host-interface.

(6) A netlink message associated with the previous step is received at portsyncd, which is silently discarded as all SONiC components are by now fully aware of the port-down event.

Port state interactions (2)



(0) During initialization, portsyncd establishes communication with the main DBs and subscribe to port/link-state netlink channel.

(1) portsyncd starts by parsing the port-configuration file (port_config.ini) associated to the hardware-profile/sku being utilized. Port-related information is transmitted to APPL_DB.

(2) Orchagent hears about it but defers acting on it till portsyncd is fully done parsing port_config.ini information. Orchagent then proceeds with port-interfaces initialization by invoking sairedis APIs.

(3) Syncd receives this new request through ASIC_DB and prepares to invoke the SAI APIs required to satisfy Orchagent's request.

(4) Syncd makes use of SAI APIs + ASIC SDK to create kernel host-interfaces associated to the physical ports being initialized.

(5) Previous step triggers a netlink message that is received by portsyncd. Upon arrival to portsyncd of the messages associated to all the ports in port_config.ini (in step 1), portsyncd will proceed to declare the 'initialization' process completed.

(6) As part of the previous step, portsyncd writes a record-entry into STATE_DB for each of the ports that were successfully initialized.

(7) From this moment on, applications previously subscribed to STATE_DB content, will receive a notification (green-light).

SONiC Modularity
FRR example

SONiC build-infra overview

- All SONiC artifacts are built within (yet another) docker container: “build-slave”.
- “sonic-buildimage/Makefile” acts just as a wrapper to build the “slave” container and to proxy “make” instructions accordingly.
- SONiC’s real makefile is sonic-buildimage/slave.mk.
- “slave.mk” processes all targets defined within “sonic-buildimage/rules/”, which incorporate all building-recipes.
- “sonic-buildimage/dockers/” holds all dockers infra files: Dockerfiles, entrypoints, supervisord, etc.
- Conditional compilation flags defined at “rules/config”.

A new docker image -- FRR example

- New ”rules/docker-fpm-frr.mk” file. SONiC specific Instructions to build FRR packages are provided here.
- New “dockers/docker-fpm-frr/” folder with Dockerfile and feature-specific components (entrypoints, process-managers, default-configs, etc).
- New ”systemd” entry in “files/build_templates/” to integrate new docker-image within SONiC process-manager.
- Build new docker image: “make target/docker-fpm-frr.gz”

Upload new docker image -- FRR example

- We are assuming that user wants to switch routing-stacks on the fly. None of these steps are needed if activating FRR at build-time.
- Typical docker-management instructions:
  - Import image into dockerd engine: “docker load < docker-fpm-frr.gz”
  - Stop previous Quagga container: “docker stop bgp”
  - Remove previous container: “docker rm bgp”
- In this particular case, we want to adjust “bgp” systemd’s initialization script to point to the new FRR image instead of the default Quagga one:
  - “sudo sed -i 's/docker-fpm-quagga/docker-fpm-frr/' /usr/bin/bgp.sh”
- Launching FRR service/docker:
  - “sudo service bgp restart”

SONiC CLI Overview

SONiC CLI overview

- Natural and intuitive – full interaction with native linux shell.
- Python based – relying on python's Click module: http://click.pocoo.org/5/
- Placed within the host-system at "/usr/lib/python2.7/dist-packages/"
- Example: https://github.com/Azure/sonic-utilities/blob/master/show/main.py#L918
- Flexible / Customizable: https://github.com/Azure/sonic-utilities/blob/master/show/main.py#L572

Thank you