Deep dive into SONiC Architecture & Design

Unlike traditional and proprietary network operating systems, SONiC – Software for Open Networking in the Cloud – has emerged as a groundbreaking open-source network operating system (NOS) designed to redefine how network infrastructure is administered. Among the reasons behind its popularity are its capabilities to provide L2, L3 & L4 functionalities. Its modular structure is based on Debian Linux, and the SAI – Switch Abstraction Interface – layer which essentially acts as a translator between the silicon and the software. It makes SONiC a disaggregated NOS that has the ability to work with multi-vendor silicons reducing vendor lock-in and giving a glimpse into the future of networks.

SONiC stands out as a complete Network Operating System. The architecture comprises a collection of software components & tools that work in harmony to offer serviceability, extensibility, development agility, and resource efficiency. All major subsystems are enclosed within Docker containers and facilitate Inter-Process Communication within the centralized system.

Application Containers
SONiC architecture employs user-space services within containerized modules, each serving specific networking functions. These services include FRR routing stack (BGP container) for routing, LLDP for topology mapping & device discovery, SNMP server (based on NGINX) for monitoring & configuration management, and PMON for hardware management & sensor monitoring. The Teamd container ensures network robustness, managing LAG groups and link status for redundancy. The DHCP-Relay container facilitates DHCP communication between clients and servers, ensuring IP assignments and configuration. Modularity allows tailored configurations to meet network needs.
Infrastructure Containers

Switch State Services (SWSS), Database, and Synchronization daemon (SyncD) serve different roles in a SONiC as compared to application containers. They are responsible for the fundamental management and configurations like switch configuration, Port updates, ASIC management & maintaining key-value configurations.

Database – REDIS

Database container hosts the redis-database engine. Redis – an in-memory database serves as a critical datastore for SONiC components, enabling efficient storage and retrieval of network states, configuration, and operational data in key-value pairs. The redis-server exposes a unix socket & binds to the host filesystem allowing SONiC components to access the database. The database container maintains several databases to organize data according to its nature and use, such as Application, Configuration, State, Counter, and ASIC-specific DBs.

Switch States Services

SWSS functions as the central communication hub for SONiC's modular components, serving as a bridge between the high-level configuration and control interfaces of SONiC, and the underlying switch hardware. Its primary role is to maintain state consistency among various modules and applications. SWSS connects to the centralized Redis engine to retrieve and update data that is stored in key-value pairs. This includes data related to VLANs, ports, routes, etc. Orchagent is an essential component of the SWSS container, translating configurations from the APPL_DB as per the SAI API and publishing them into the ASIC_DB. These instructions are communicated with the switch's ASIC driver via the SAI interface to apply configuration changes and update the switch hardware accordingly.

Synchronization Daemon – The way to talk to hardware!

SAI – Switch Abstraction Interface – is designed to be vendor-agnostic, which means SONiC can work with a wide range of switch hardware without having to undergo extensive modifications or adaptations. It acts as an intermediary between SONiC and Hardware-specific switch ASICs (Application Specific Integrated Circuits). The primary role of Syncd is to synchronize and push network state & configuration changes to the hardware and simultaneously collect real-time information from the hardware. Its role involves setting up and configuring ASICs to match the desired network configuration, including data plane forwarding, switch port configuration, VLANs, and other networking functionalities. SyncD leverages the SAI API to interact with the hardware components using ASIC SDK. Hardware vendors typically provide standard SAI interfaces, implemented using vendor specific SDKs (Software Development Kit) required to drive their ASICs. These drivers are essential to harness the full capabilities of the underlying hardware as they enable hardware acceleration and optimization, improving network performance and reducing the load on the main CPU.

Linux Kernel

SONiC leverages linux kernel capabilities like creating **Network namespaces**, separate network stacks with their own interfaces, routing tables, firewall rules, and settings to achieve network isolation, useful for managing multiple ASICs), **Inter-Process Communication** mechanisms which SONiC uses for communication between different network services and daemons, enabling seamless cooperation among SONiC components, **ACL** functionalities like firewalling, packet mirroring, Policy Based Routing (PBR), and control plane ACL, and **Switch memory management**, including network-specific functionalities, such as managing network configurations, Quality of Service (QoS) policies, packet forwarding, and other switch-related tasks. SONiC defines and controls how memory is allocated and used for networking purposes, including switch memory management for QoS maps, shared buffers, policers, schedulers, and queueing algorithms. It also integrates various features related to switch memory management, like Weighted Random Early Detection (WRED) and Explicit Congestion Notification (ECN). It supports NAT tools like SONiC supports NAT, tools like iptables and nftables . SONiC can use Berkeley Packet Filter (BPF) to filter and selectively process Netlink messages. BPF filter programs can be attached to specific Netlink sockets in the kernel, and they are executed for each incoming Netlink message, deciding whether to accept or drop the message.

Configuration Management

There are several ways to configure the SONiC NOS via the SONiC-CLI, REST-API and minigraph. Free Range Routing (FRR) CLI for routing configurations. The SONiC-CLI, which is based upon a CLICK Python library, provides a SONiC-centric and user-friendly way to interact with the operating system, making it easier to configure network modules, view status information,

troubleshoot network settings, and access operational data such as interfaces, routing tables, and system logs.
With SONiC, you can use SSH to log in to the CLI from a remote location. The REST API in SONiC offers a programmatic way to interact with the network operating system, following REST principles and using standard HTTP methods like GET, POST, PUT, and DELETE. It employs a client-server architecture for seamless communication. Additionally, YANG models play a crucial role in SONiC by defining network configurations and operational data. SONiC also uses a JSON file named “config_db.json” to store configuration data. This file can be edited to configure various aspects of the NOS, useful for making bulk configuration changes or for scripting configuration updates.

Conclusion

SONiC, a transformative NOS, revolutionizes network management with its modular, vendor-agnostic design. It offers broad hardware support,and programmability through user and kernel space, enhancing network performance and scalability. SONiC’s open-source, containerized architecture empowers network operators to tailor their infrastructure, fostering automation and adaptability in the dynamic data center landscape. It represents a future where open-source, extensible solutions simplify modern networking for administrators.