Switch Abstraction Interface (SAI)

A Reference Switch Abstraction Interface for OCP

Executive Summary

Switch Abstraction Interface (SAI) defines an abstraction interface for switching ASICs. The interface is designed to provide a vendor-independent way of controlling both switching entities like hardware ASIC's or NPU's as well as software switches in a uniform manner. This specification also allows exposing vendor-specific functionality and extensions to existing features.

Something as basic as getting switches to forward packets requires implementing multiple management protocols and configuration generation which is undifferentiated work. Even basic routing protocols are now undifferentiated work. SAI allows the same network software stack to program and manage many different switch chips without undergoing any changes.

SAI helps to easily consume the latest and greatest hardware when we can run the same application stack on all our hardware, enabled by a simple, consistent programming interface**.**

SAI helps us to keep the base router platform simple, consistent, and stable. Thus shifting our focus to applications that require integrating our network with our cloud. We believe that fulfills a necessary part of the software ecosystem and is a big step towards open networking software.

Contents

Figures

Revision History

Name	Date	Version	Description
Kamala Subramaniam	2015-03-28	0.1	Initial Release
Kamala Subramaniam	2015-07-07	0.2	Added a clause that all maintainers/contributors must be an OCP member.

Overview

Switch Abstraction Interface (SAI) is a standardized API that allows network hardware vendors to develop innovative hardware architectures to achieve great speeds while keeping the programming interface consistent. SAI helps easily consume the latest and greatest hardware by running the same application stack on all the hardware, enabled by a simple, consistent programming interface**.** New applications can run easier and faster on the latest hardware with lesser portability of bugs.

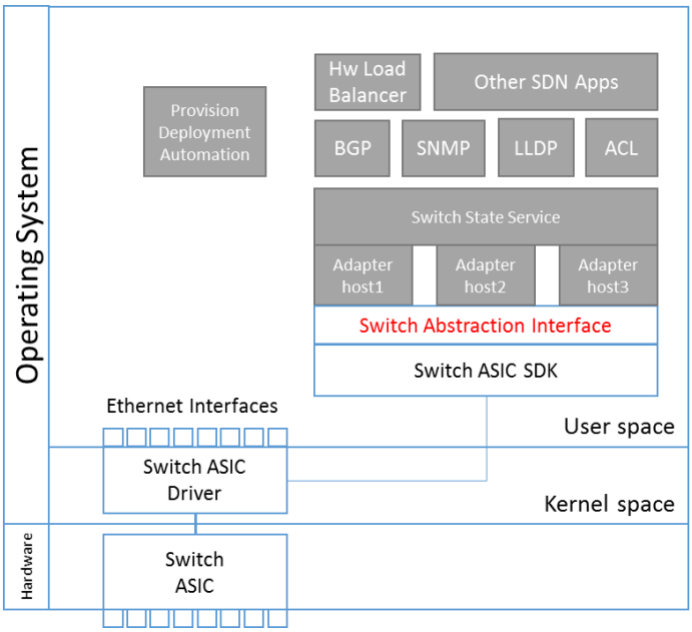

Figure 1: SAI in a plausible switch system architecture

Starting off we had three main goals:

1. The ability to develop SAI on different vendor platforms.
2. Demonstrate the ease of implementation of a Layer 3 IP router
3. To deploy a SAI implementation in an operator's network.

It's been great to see tremendous support from several industry contributors to the SAI v0.92. Contributions to the OCP community for the SAI initiative are submitted as proposals to the OCP Networking SAI GitHub repository. Following a proposal submission, are discussions on the OCP mailing list that further define the proposal. Eventually the code is also submitted to a GitHub repository. The SAI v0.92 introduces numerous proposals including:

- Access Control Lists (ACL)
- Equal Cost Multi Path (ECMP)
- Forwarding Data Base (FDB, MAC address table)
- Host Interface
- Neighbor database, Next hop and next hop groups
- Port management
- Quality of Service (QoS)
- Route, router, and router interfaces

License

Background

Today in our production networks, we deploy a multitude of vendors across many layers, be it ASIC manufacturers or switch vendors. The reality is that something as basic as getting switches to forward packets requires undifferentiated work. Be it basic Access Control List (ACL) rules, managing the Routing Information Base (RIB), programming chips, and managing its Forwarding Information Base (FIB). Even basic routing protocols are now all undifferentiated work. Yet, the underlying complexity of the hardware, and the strict coupling of protocol stack software to the hardware, denies us the freedom to pick and choose the combination of hardware and software that is the best suited for our networking needs.

SAI helps us to keep the base router platform simple, consistent, and stable. It also reduces the time to market, and adopt the latest available hardware. It breaks the software-hardware coupling and enables us to choose the best fit of software and hardware on a need by application or a need by network base. By providing simple, consistent interfaces for applications and protocol stacks that orchestrate and automate cloud services, it helps consume the underlying complex and heterogeneous hardware easily, thereby shifting our focus to applications that require integrating our network with our cloud. Switch Abstraction Interface (SAI) is therefore a big step towards open networking software.

Additionally, the evolution of Software Development Kits (SDKs) cracked the wall a little between software and hardware. But only in as much to make the chips more programmable by the applications. SDKs are APIs commonly written in simple C like functions enabling the applications/protocol stack access to the switching ASIC. However, different switching vendors will have different SDKs serving as a wrapper for their proprietary algorithms. SAI breaks this wall by not just being another SDK wrapper but by being a standardized API.

A standardized API allows network hardware vendors to develop innovative hardware architectures to achieve great speeds while keeping the programming interface consistent. As new hardware functions are exposed, hardware vendors can introduce extensions to the API. Revisions to the baseline standard could occur. This would introduce change at the hardware programming level, but this change would be much less frequent than today as it would only be required for functional changes, not simply implementation differences.

Design

SAI consists of the following modules:

Adapter is a pluggable code module, supplied by either a vendor or control plane stack owner, that contains either ASIC SDK code itself or client module for ASIC SDK hosted in external process and implements the interfaces described in this specification; for all practical purposes it is equivalent of a “user-mode driver”.

Adapter host is a Microsoft or vendor-supplied component that loads the adapter and exposes its

functionality to the control plane stack.

Switching adapters are user-mode drivers, typically supplied by ASIC vendors. Adapters are registered with the switching stack and then can be loaded as needed. It is a responsibility of the adapter to discover and bind to the specified underlying hardware, including loading of or attaching to kernel-mode drivers if needed.

Adapters are expected to be as simple as possible, ideally simple wrappers around vendor's SDKs. Our design strives to push the bookkeeping complexity from adapter into the adapter host wherever possible.

The adapter module is loaded into a hosting process ("adapter host") and then initialized. During initialization the adapter initiates discovery process of the specified instance of a switching entity. A switching entity is a top-level object in this API.

SAI API:

The SAI interface is a local interface between the Adapter Host and Adapter. ASIC functionality is exposed to the rest of the system by the Adapter Host through other mechanisms, which are not part of this specification.

The API is designed to be platform-agnostic (*nix/ Windows/etc…).

The API is attribute-based to minimize compatibility issues with versioning of structures and to allow API extensibility.

The API is a collection of C-style interfaces exposed from the adapter. These interfaces are grouped into three categories:

- Mandatory functionality. This is a set of "core" interfaces which are required to build a basic routing appliance. All vendors must support these interfaces and the switching stack will fail loading of the adapter if any of these interfaces are missing.

- Optional functionality. This is a set of additional interfaces which are not required, but enable various scenarios in a modern datacenter. Definition of these interfaces is common for all the vendors. These interfaces are not required by the switching stack to be exposed from the adapter. However, they become required if a given system configuration references any of these features.

- Custom functionality. This is a set of interfaces that are unique for a vendor and is not standardized. The default adapter host is not aware of these interfaces and a custom adapter host can be supplied by the vendor to expose these interfaces to the switching stack. We also intend to propose a more generic framework for exposing such functionality through default supplied adapter host in further drafts

Test Plan

The test plan is still being worked. Multiple options are being looked at such as using UNH for testing resources, Google test framework, and Barefoot's P4 software emulator. The test_sai and build_sai applications that we are looking at writing will provide levels of SAI compliancy. As the SAI program grows, we expect this testing program will increase substantially.

Checklist for Maintenance

Currently the code is maintained in GitHub and the development uses GitHub-based best practices. All code changes are reviewed publicly (using GitHub's online code review tools) and approved by someone with commit rights. The current list of committers/maintainers includes:

- Microsoft
- Dell
- Mellanox
- Broadcom
- Cavium

It is mandatory that all entities (including the ones listed above) with code approval or commit capability, i.e., are either committers/maintainers into the SAI project be OCP members. We are open to expanding the committers list as other contributors/authors emerge. New contributors/authors cannot become committers/maintainers without first being an OCP member.

In the event that all maintainers are permanently unavailable, a duly appointed representative of the Open Compute Project may take over the project.

Software releases will be made as time and major features are committed. While many open source projects with regular committers have a time-based release model, at least for the near future until the projects popularity increases, we will follow a feature-based release schedule.

Checklist for Governance

This is the list of current governance sites which may change with acceptance into OCP.

Website: N/A

Mailing list: opencompute-networking@lists.opencompute.org

IRC: N/A

Mirror: N/A

GitHub: https://github.com/opencomputeproject/OCP-Networking-Project-Community-Contributions

Wiki: http://www.opencompute.org/wiki/Networking

Roadmap

There are three different future directions that make sense with SAI: building applications for the SAI, building an open control stack on top of the SAI, and architecting SAI to work on multi-chip/multi-chassis platforms.

Building applications for the SAI:

- build_sai, test_sai, debug_sai, etc.

Control Stack

- Build a Control Plane Stack (CPS) that works in conjunction with the SAI to make it vendor agnostic is the truest sense.

Multi-chip/Multi-chassis:

- Architecting SAI to work on multi-chip/multi-chassis platforms

Supporting Documents

The majority of the technical documents live in the GitHub directory in the source, including:

- SAI architecture
- Individual proposals
- SAI code