Network OS Models

Dinesh G Dutt

What has an OS got to do with networking?

Why should I care?

The OS you pick largely determines the kind of network you build

Courtesy of https://www.flickr.com/photos/usdagov/41801154804

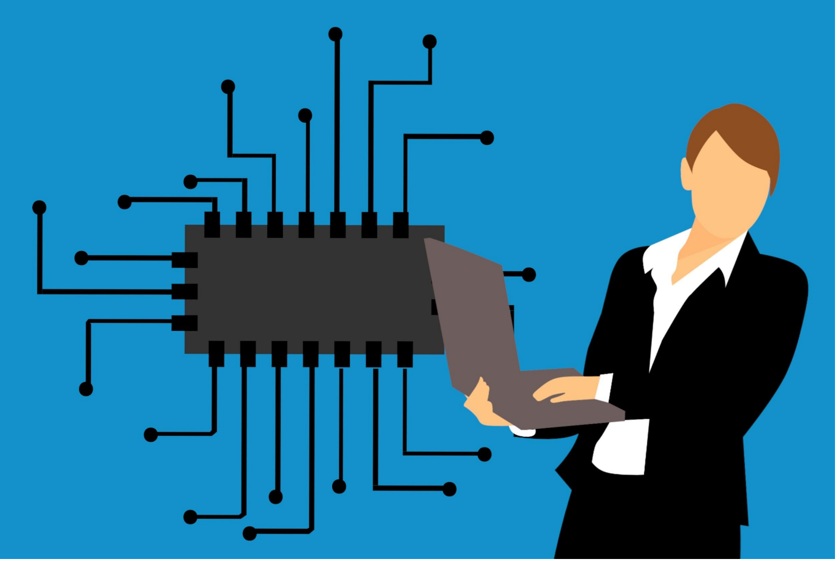
Fred runs a small-ish network

He is responsible for 200 servers and about 16 switches. He is responsible for compute, storage and network. He needs to have a simple, consistent workflow to manage this entire datacenter.

Courtesy of
https://www.flickr.com/photos/seeweb/26191054733

Rama is a network architect

He works for a large DC operator and is responsible for the design and smooth functioning of the network. His network management needs are in a class of their own. Off the shelf tools just don’t cut it for his network.



Courtesy of https://pxhere.com/en/photo/1448279

Maya is designing a new switching chip

She’s the founder of a startup that is working on a programmable switching chip that has much higher speeds and feeds. She needs to demonstrate her chip to the datacenter operators.

Courtesy of Wikimedia

Radia is designing a new routing suite

She’s working on an open source routing suite that has many useful innovations.

Courtesy of Wikimedia

Joe operates a network at a mid-size enterprise.

He’s been tasked with figuring out how to play in the new networking landscape. He cares about having the least amount of disruption: in the network, in his work life.

The OS in each case largely determines if their needs can be met

Who Am I?

Dinesh Dutt

Engineer, Author, former Chief Scientist, former Cisco Fellow

Opinions are mine! I don’t work for anyone :)

The key takeaway is an understanding of how the OS you choose affects the network you can build

—

We'll cover NOS models, switch ASIC programming models, design and tradeoffs of real-life NOSes in this talk

The Driver

The rise of modern data center ignited a **REVOLUTION in NETWORKING** fueled by network operators against network vendors

Courtesy of https://en.wikipedia.org/wiki/Revolution

The issue: cost

- → **CapEx**
  The high cost of network equipment, and especially cables

- → **OpEx**
  The high cost of managing a network, especially at scale

Courtesy of http://www.thebluediamondgallery.com/wooden-tile/c/cost.html

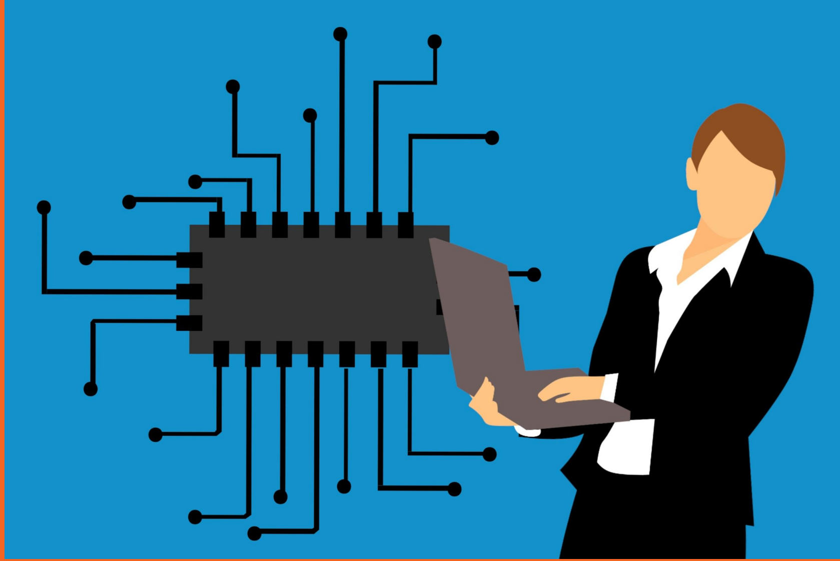
A fundamental difference between modern data center and the past: The focus on operational efficiency* and pace

* Cost is the fundamental driver

Fred & Rama care about **programmatic access** and a **uniform workflow** between compute, network & storage

Radia & Rama care about the ability to replace vendor code with third-party software

Maya cares about having the OS support a new switching silicon ASAP



Joe's primary concern is stability, avoiding vendor lock-in, and minimal disruption to status quo

This focus leads to the requirements

- Programmatic access to the device
- Run third-party applications
- Replace vendor applications with third-party equivalents
- Operator can fix bugs
  - This is a requirement largely for hyperscalars
- Avoiding vendor lock-in

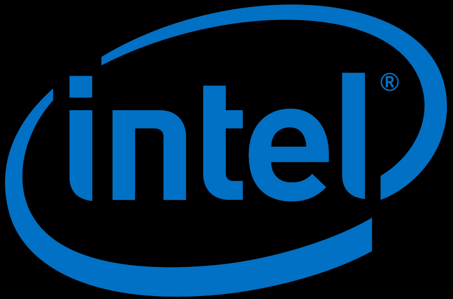
Ask these technical questions for each option

- Does the choice make the device easier to program?
- Does the choice allow you to run any network-based third-party application?
- Does the choice allow you to replace a vendor-supplied software with another one?
- Does the choice allow easier addition of new switching silicon?
- Does the choice enable greater engagement?

Single business question of any choice: how does the choice help or hurt running my business?

The Players

Ubiquitous Technologies



For low-end devices

Examples of NOS Players

ARISTA
CISCO™
CUMULUS
JUNIPer NETWORKS
SONiC

The size of the logo is an indication of my patience and skill at trying to make them all look uniform

Just about every network OS today uses the Linux kernel*

* I hear JunOS is shifting gears too

Benefits of Common Infrastructure

“Google uses a very common set of building blocks across all of its software, so by instrumenting these building blocks Dapper is able to automatically generate a lot of useful trace information without any application involvement. “

- Dapper Paper, 2015

Why more than one model?

Linux is common, but the variations are in:

- Location of switch network state
- Handling of the switching silicon driver
- Use of the Linux kernel network stack
- User/Application interface model

Network Switch State

Network switch state is the primary source of truth of the state of the networking components on the device

Includes interfaces, mac table, VLAN, routes, ARP/ND table, VRFs, tunnels, etc.

Network switch state can reside either primarily in user-space or in the kernel

Most common model is to maintain network state in NOS-specific user space

Examples include Arista’s EOS, Cisco’s NXOS, SONIC etc.

Storing network state in Linux kernel is vendor-agnostic

Example is Cumulus Linux

Linux kernel networking stack is just regular networking stack: routing, bridging, tunnels work the same as on any traditional vendor.

The **main implication** of where network state is stored is in the **application model**

Single well-defined open application API or vendor-specific

Similar to how it was in the Unix-land in pre-Linux times

Most use cases won't "just work" if state is in vendor-specific user-space

Think telemetry agents, configuration agents...
Think routing suites, open & closed
Think languages
Think customizability
Think pace of innovation and business agility

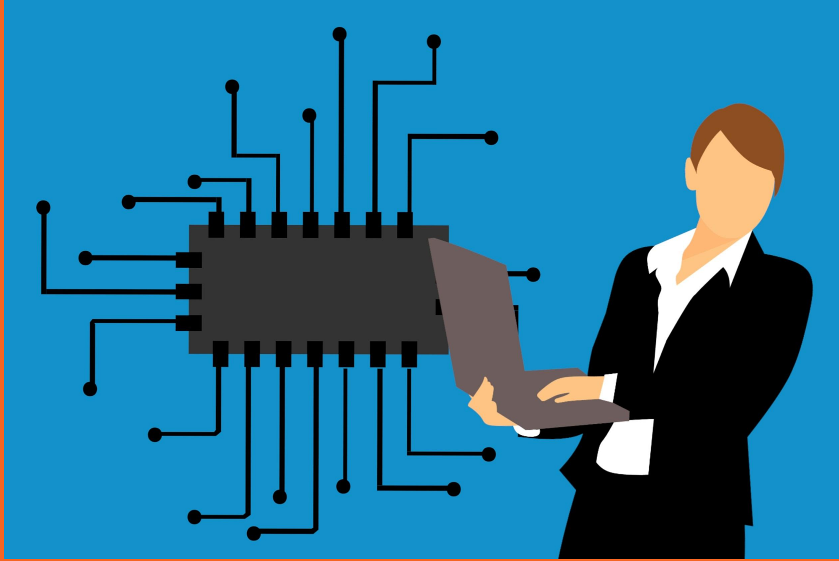
The needs of Rama, Joe, Fred & Radia aren’t served if network state is in vendor-specific blob

Joe gets stability if state is in Linux:

Access to mature, sophisticated tools to more easily ensure a stable network

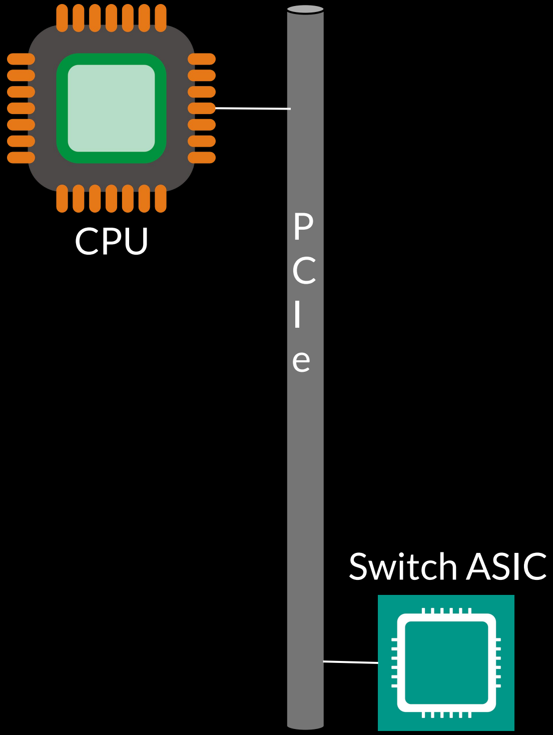
Stability because there are more eyes on the Linux kernel than any single vendor’s

But does storing state in kernel make it easy for Maya’s needs to be met?



Switching Silicon-OS Interaction

Switch ASIC - CPU Connection



- Switching ASIC & CPU are typically connected via PCIe or similar bus.
    - 10-40G bandwidth
- Some ASICs provide connectivity via dedicated Ethernet ports

Icons are courtesy of pixabay

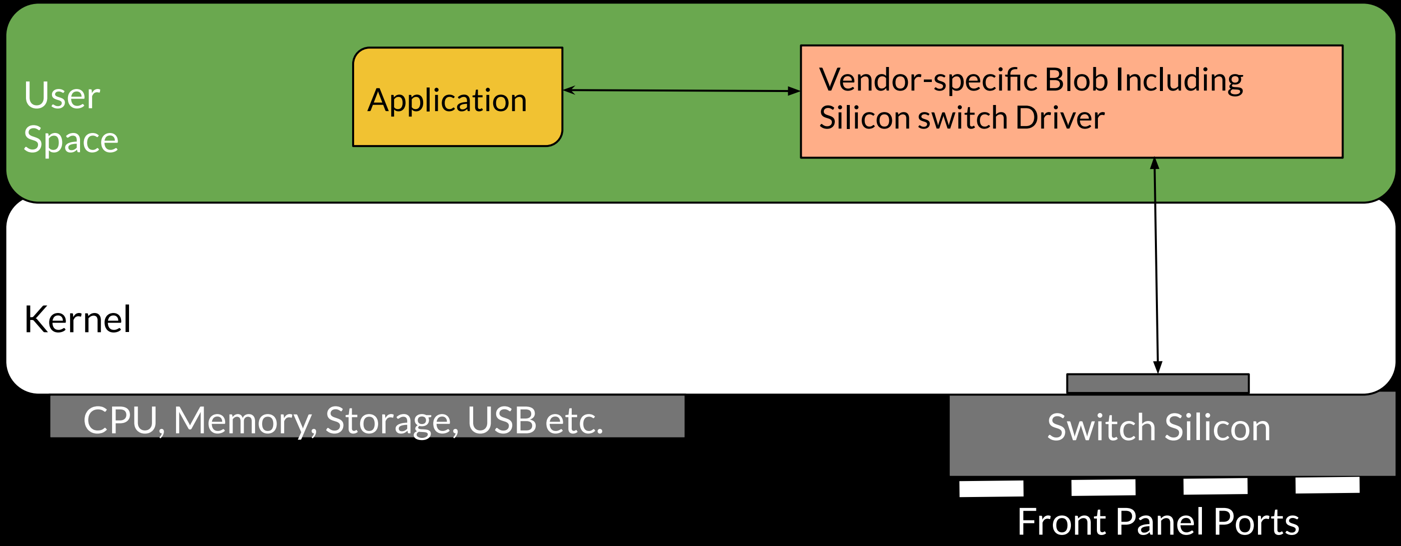
Questions about switch-cpu interaction

- How are switch ports represented in the OS?
  - Do they even show up?
- How are packets sent from the CPU delivered to the right outgoing front panel port? I.e. Packet Tx/Rx Path
  - And vice-versa
- How is the switching chip programming done?
  - Includes programming routes, interface settings, mac addresses etc.

Possible Answers to these

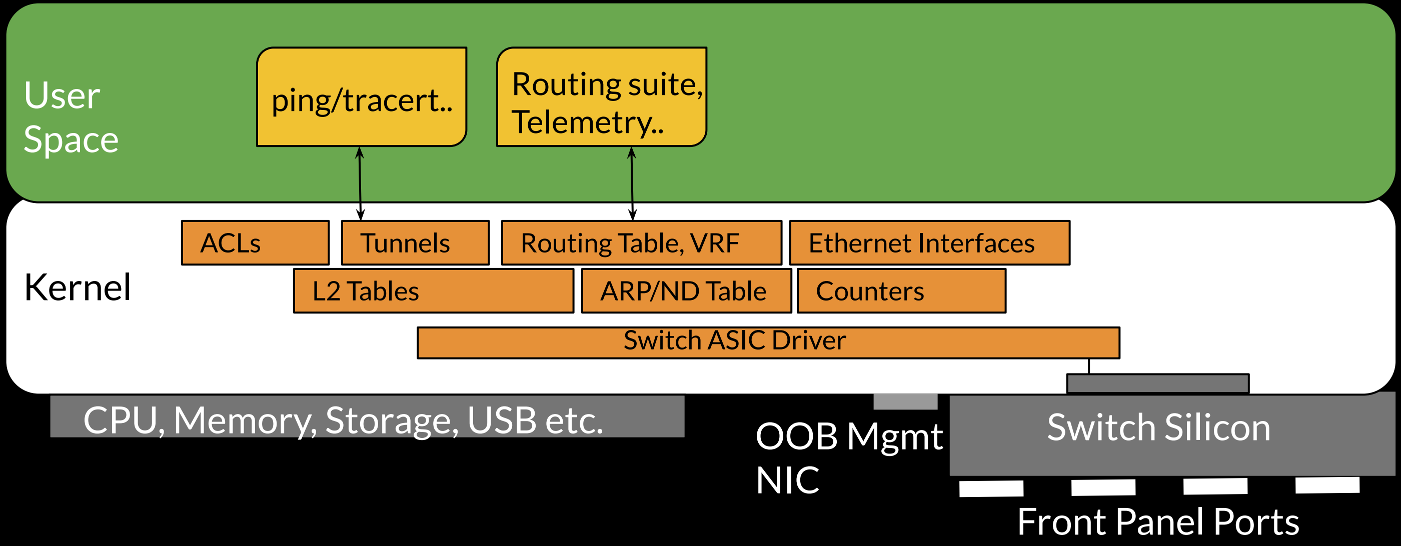
1. Nothing is in the Linux kernel, everything is in userspace
2. Everything is in the Linux kernel
   a. Only the switch ASIC driver is in the user space
3. Hybrid between the first two, with limited use of the kernel

(1) Switching Silicon is Invisible to the OS

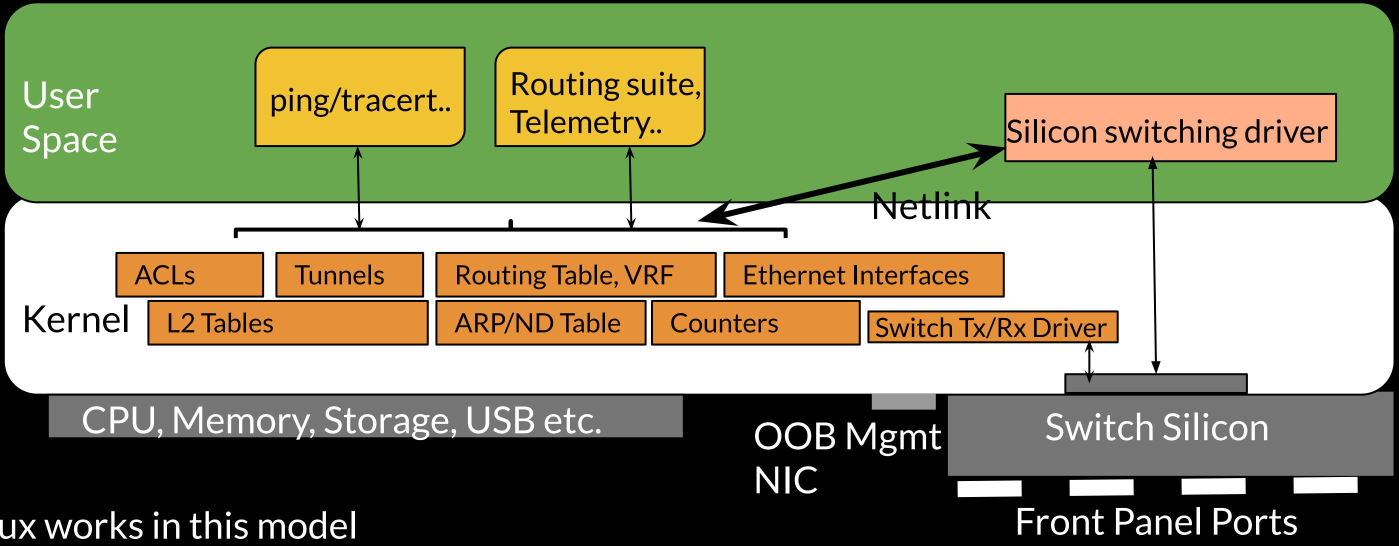

User Space
Application
Vendor-specific Blob Including Silicon switch Driver
Kernel
CPU, Memory, Storage, USB etc.
Switch Silicon
Front Panel Ports

- NXOS follows this model
- No network application can run unmodified.

(2) Everything is in the Kernel

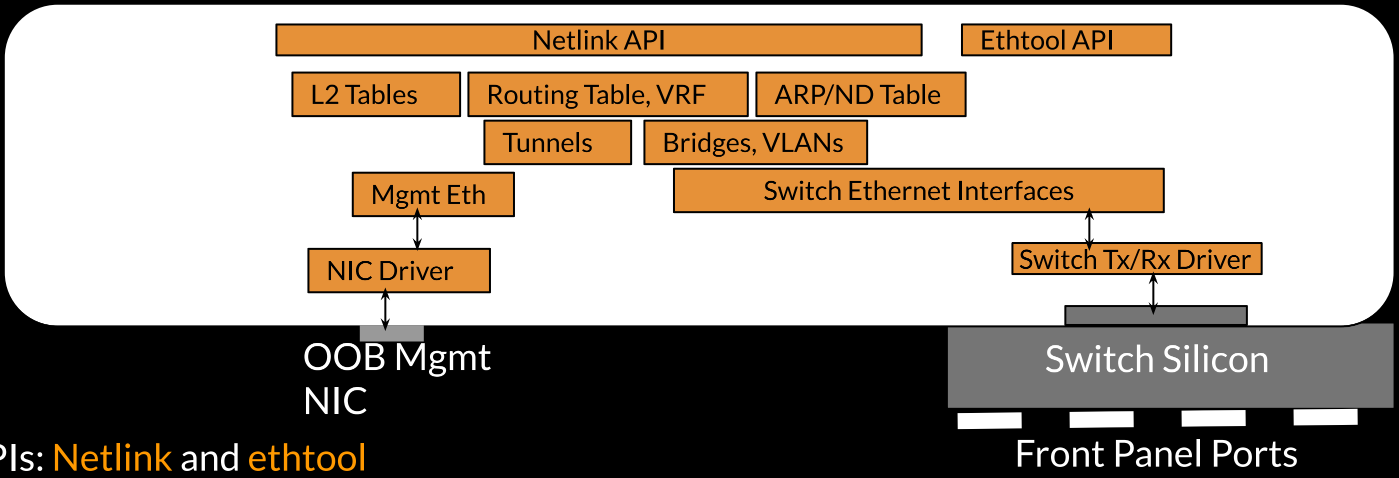


(2a) Everything Except Driver is in the Kernel



- Cumulus Linux works in this model
- Linux applications can run unmodified because they only interact with native Linux kernel API

Decomposing Linux Kernel Network API



- Two main APIs: Netlink and ethtool
- Ethtool is for physical link management (speed, duplex, buffers etc.)
- Netlink is for the rest
- Every network device is represented as a netdev object in the kernel

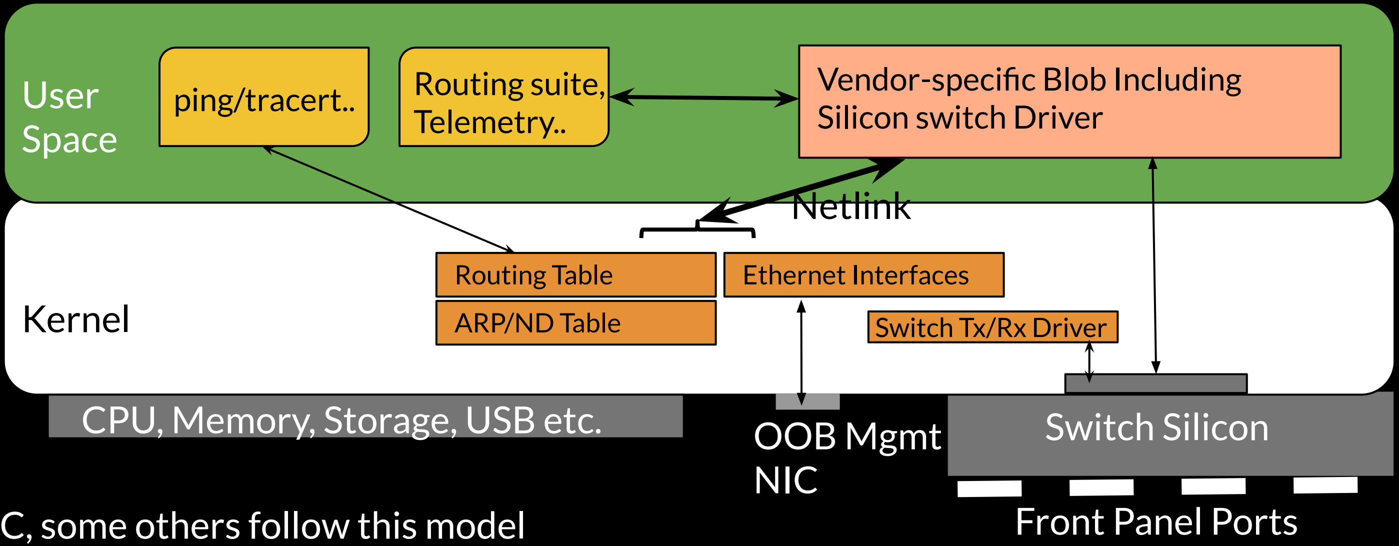
Netlink

Netlink is the Linux kernel API that is used to:

- Update Interface, Address, Route, & ARP/ND state
- Notify listeners on any successful change to any of the above states
- Has bindings in multiple languages
- Defined in RFC 3549, but evolved a lot since

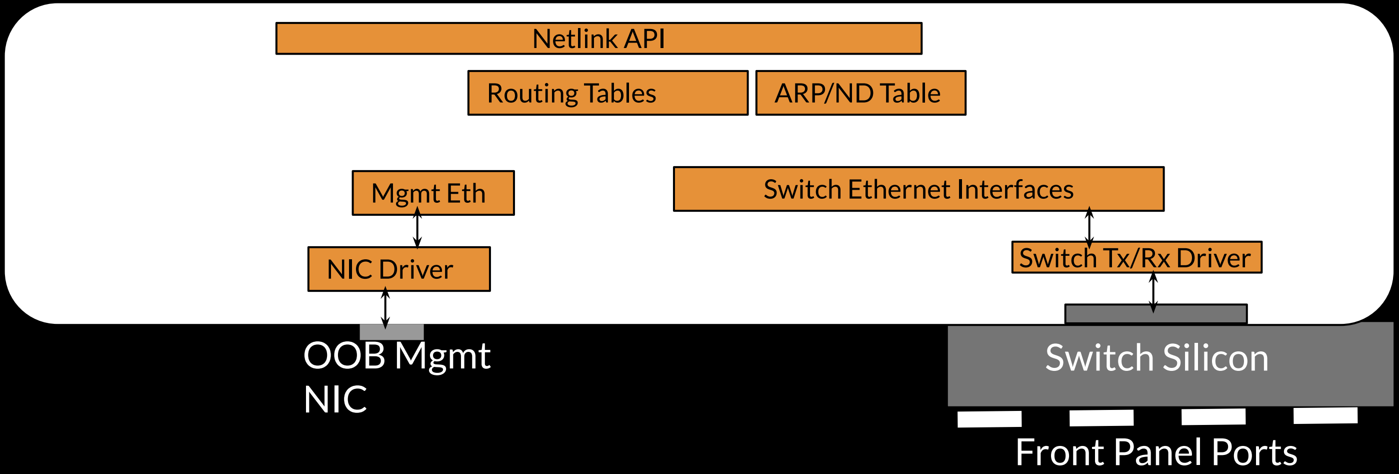
Netlink API uses socket interface

(3) Hybrid Model Application Model



- Arista, SONIC, some others follow this model
- Switch silicon ports appear as regular ethernet interfaces
- IP routing tables are populated/sync'd between vendor blob & kernel
- Some applications can run unmodified, cannot change network state

(3) Hybrid: Partial State Sync with Kernel

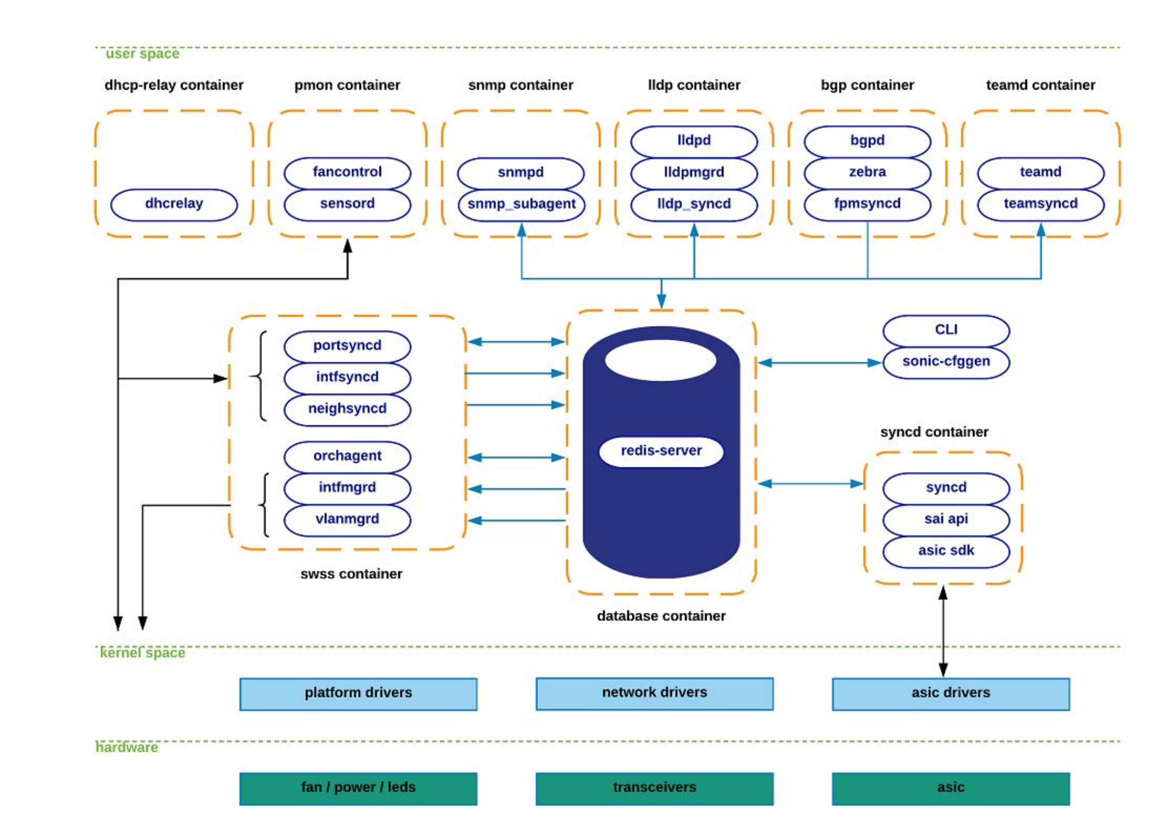


What are the benefits/limits of partial sync?

Network apps that don’t modify state work
Network apps that modify only sync’d state work

Configuration agents etc. can’t run unmodified
Telemetry agents can’t run unmodified unless counters are sync’d

Exploring Hybrid Model: SONIC Architecture



Courtesy of https://github.com/Azure/SONiC/wiki/Architecture

SONIC Architecture

State is stored in redis DB

SWSS responsible for storing state in redis based on events from various modules including control protocols & Linux kernel netlink messages

Separate syncd container to interface between silicon and state in redis, uses SAI

From a model perspective, SONIC uses Linux kernel with partial state sync, similar to Arista & some other NOS

In EOS, apps that require tunnel encap/decap work as long as IP routing tables and ARP tables are correct. ASIC does tunnel operations

EOS VM image* uses Linux's TUN/TAP devices with user space packet forwarding

* VM image uses different packet forwarding than physical image

Programming Switching Silicon

How do you translate kernel network state into hardware?

Unlike NICs, USBs and disks, switch ASIC had no device independent abstraction.

So far, every vendor has had their vendor-specific switch programming software: running in user space

Arista, Cumulus, NXOS etc. all have this same model

Two models proposed recently for a generic switch ASIC abstraction on a NOS: SAI and switchdev

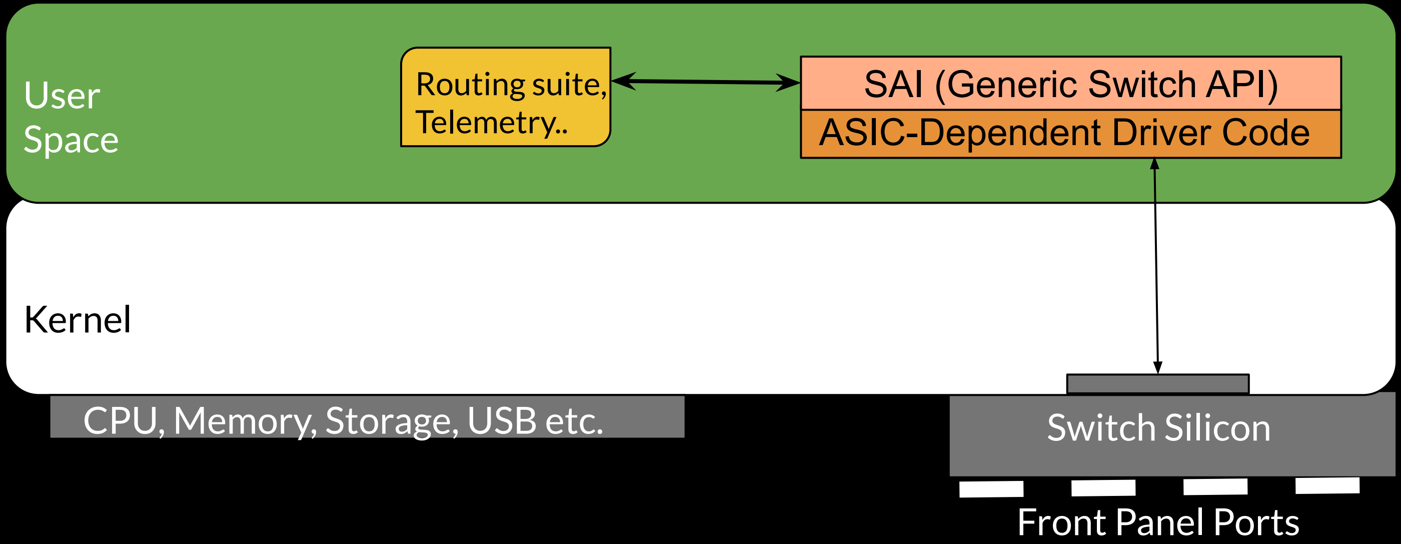
SAI & Switchdev

Switch Abstraction Interface(SAI)

https://github.com/opencomputeproject/SAI

- Proposed by Microsoft/Dell/Mellanox, adopted by OCP
- Limited feature set: for Microsoft-specific Azure network only
- Economic gravity: required to play at Microsoft

SAI Model


User Space
Routing suite, Telemetry..
SAI (Generic Switch API)
ASIC-Dependent Driver Code
Kernel
CPU, Memory, Storage, USB etc.
Switch Silicon
Front Panel Ports

- SAI provides a vendor-independent, device-independent switch abstraction API
- Switch ASIC vendors provide their driver code mated to SAI API
- Allows silicon switch programming software to be ASIC independent by writing to SAI

SAI Features Supported

Using https://github.com/Broadcom-Switch/SAI and https://github.com/Mellanox/SAI-Implementation, SAI features supported are:

- Basic IP routing
- Basic Bridging including VLANs
- LAG
- Basic QoS including Priority Flow Control (PFC)

SAI Platforms Supported

- Broadcom: Trident2, Tomahawk & Tomahawk2
- Mellanox: Spectrum
- Also heard Barefoot’s Tofino is supported

Switchdev

- Native Linux kernel abstraction for a switch ASIC
- Proposed by Mellanox and Cumulus
- Mellanox switch ASICs supported under this model
- Driven by engineers in Linux kernel network group

Switchdev Silicon Programming

- Uses notifiers built into every network object in the kernel
- Hardware driver registers to be notified whenever any network object changes state
  - Registers with all appropriate network object
- When any network object is updated, notifiers within the object are invoked to push updates to hardware

Switchdev does not imply silicon driver has to be GPL! Not different from proprietary graphics drivers

SAI vs Switchdev

SAI

- Silicon programming model only
- No specification of the user API for network state
- Multiple switching silicon vendors support SAI today

Switchdev

- Assumes Linux kernel network state and model
- User API is the Linux kernel network API and normal socket API
- Mellanox is the only vendor providing switchdev support today

User Interface Models

Does storing state in the Linux kernel mean bash is my only CLI?

No, you can use fish 🙂

Seriously, no.

Other infra tools, like docker & k8s, have their own CLI.

Network can do the same.

What you have today are native Linux commands such as ip, ifupdown2 and ethtool

That there isn’t an open, unified network CLI is work to be done.

One of the largest factors affecting Joe because it disrupts his status quo

Avoiding Vendor Lock-in

Exploring Open Source Possibilities

Ubuntu/Fedora* + FRR + switchdev is one possible open source NOS for whitebox switches for DC

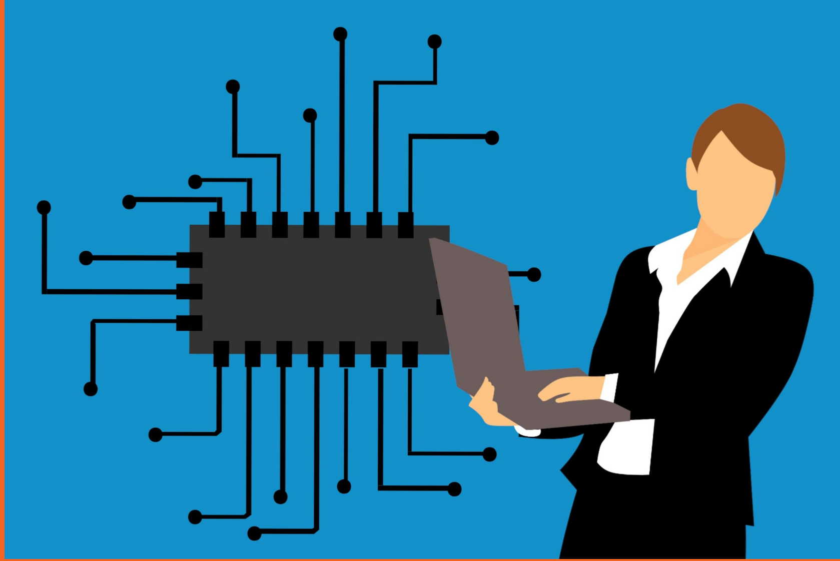
Essentially, any Linux distribution is valid here

Ubuntu/Fedora* + FRR + SAI is another possible open source NOS for whitebox switches for DC

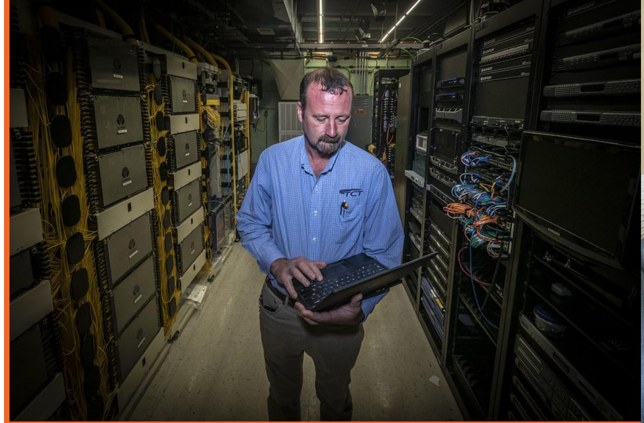
Essentially, any Linux distribution is valid here

SONIC + FRR + SAI is a third possible open source NOS model for whitebox switches in the DC

At some level, all options are viable for Maya



But what about Fred, Radia, Rama & Joe?



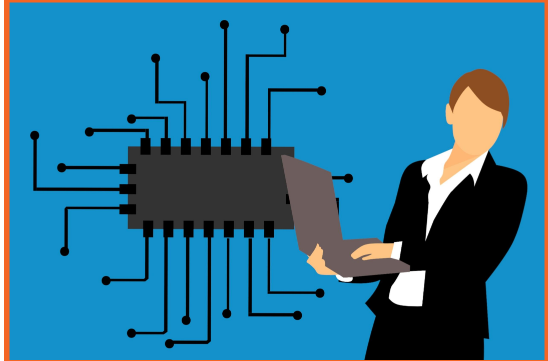
Comparing Open NOS Models

	SONIC + SAI	Linux + Switchdev	Linux + SAI
Any routing suite	No (tied to FRR)	Yes	Yes
Any telemetry	No (yes, only for SNMP)	Yes	Yes
Any config tool	No	Yes	Yes
Virtual image (as vrouter, for CI/CD etc.)	No (no packet forwarding model)	Yes	Yes
New ASIC	ASIC vendor	ASIC vendor	ASIC vendor
Customization	Depends	Yes	Yes
Who’s responsible	OCP	Kernel community	Kernel community

Comparing multiple NOS models

	Linux + Swichdev/SAI	NXOS-like Model	Hybrid Model	Linux + vendor-specific driver
Any routing suite	Yes	No	No	Yes
Any telemetry	Yes	No	No	Yes
Any config tool	Yes	No	No	Yes
Virtual image (as vrouter, for CI/CD etc.)	Yes	maybe	Yes	Yes
Platform support	ODM vendor	Vendor	Vendor	Vendor
New ASIC	ASIC vendor	Vendor	Vendor	Vendor
Who’s responsible	Kernel community	Vendor	Vendor	Vendor

Native Linux + Switchdev/SAI satisfies all the stakeholders



System validation, support etc. still need a NOS vendor*: either Red Hat-like or traditional vendor

* Unless the operator has the skill and the people to DIY

Concluding Thoughts

Using native Linux for storing network state allows the network operator the most flexibility and power.

Linux network functions just like the ones from traditional vendors

Does not require a relearning of network concepts

Avoids vendor lock-in

SAI and switchdev are WIP

Vendor-driven silicon switch programming still rules

As solely a backend, the network operations should be largely unaffected when vendor-specific driver is replaced by SAI/switchdev

FRR is fast becoming the open source routing suite of choice

@ddcumulus

https://www.linkedin.com/in/ddutt