Distributed Switch Architecture

A.K.A. DSA

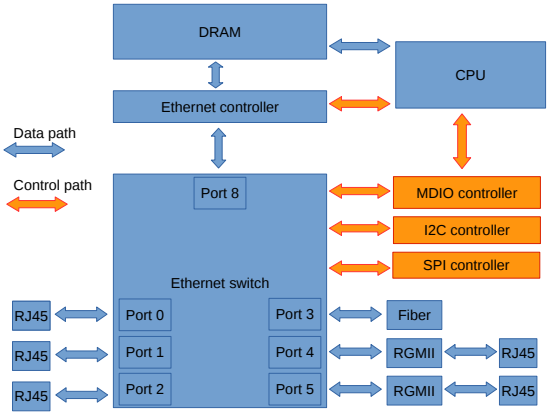
Andrew Lunn[1] Vivien Didelot[2] Florian Fainelli[3]

[1]andrew@lunn.ch
[2]vivien.didelot@savoirfairelinux.com
[3]f.fainelli@gmail.com

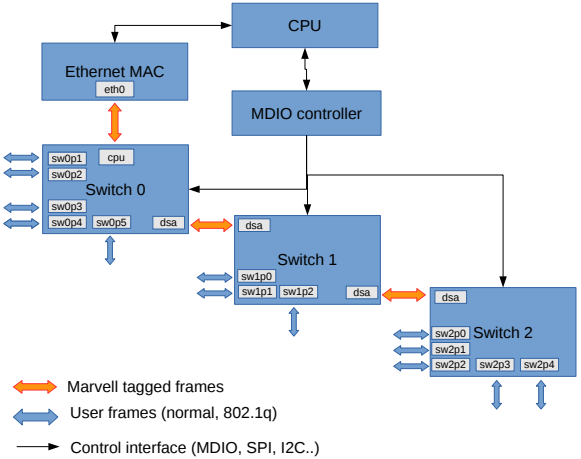
Netdev 2.1, 2017

Outline

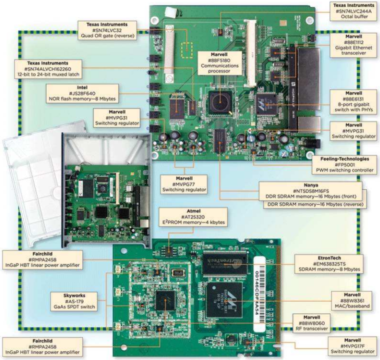
DSA in one Slide

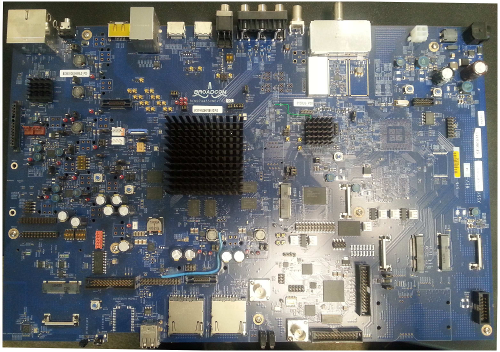


The D in DSA


CPU
Ethernet MAC
eth0
MDIO controller
sw0p1
cpu
sw0p2
Switch 0
sw0p3
sw0p4
sw0p5
dsa
dsa
Switch 1
sw1p0
sw1p1
sw1p2
dsa
dsa
sw2p0
Switch 2
sw2p1
sw2p2
sw2p3
sw2p4
Marvell tagged frames
User frames (normal, 802.1q)
Control interface (MDIO, SPI, I2C..)

Wi-Fi Access Point, Set-top Boxes



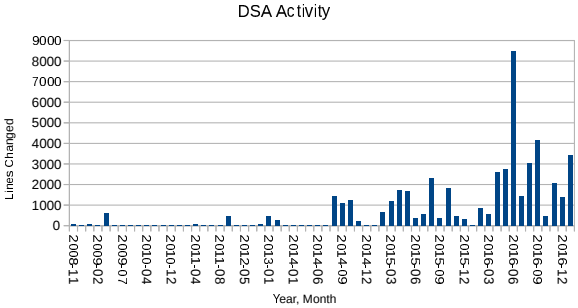


Industrial Switches/Routers, mostly Transport Industry

Supported Switches

- Marvell 88E6xxx
- Broadcom B53 (Roboswitch) and Star Fighter 2
- Qualcomm QCA8K
- Mediatek MT7530 (under review)
- Microchip LAN9303 (under review)
- WIP driver for Microchip KSZ, not yet contributed

Added to the kernel in 2008



History

2008

- First commit, support for some Marvell SOHO switches

2014

- Broadcom SF2, EEPROM, Temperature sensor, EEE, WoL, better PHYLIB integration, 88E6352

2015

- Device tree, Hardware bridging, VLANs, Fixup module unload/load, Switch reset via GPIO, netconsole

2016

- New device tree binding, Switches as Linux devices, SPI, MMIO, Broadcom B53, Qualcomm QCA8K, 88E6240

2017

- 88E6390, Second generation Starfighter 2 (BCM7278), TC offloads, port mirroring

Alternative approaches

OpenWrt/LEDE's swconfig:

- Generic netlink based configuration
- Does not make use of switch tags, uses VLAN tags for traffic segregation
- No per-port network interfaces
- Each switch driver is allowed to extend the control API: no consistency across device drivers
- Proposed as a solution in October 2013: https://lwn.net/Articles/571390/
- Discussion starting point that led to switchdev!

Other approaches

Other approaches:

- Quick-n-dirty /proc, /sys/, debugfs, ioctl() interfaces from various SoC vendors
- Vendor specific and proprietary switch SDK in user space
- Have the bootloader configure the switch!

The Switch as a Hardware Accelerator

During various conference corridor side discussions around 2014 it was decided how to model Switches

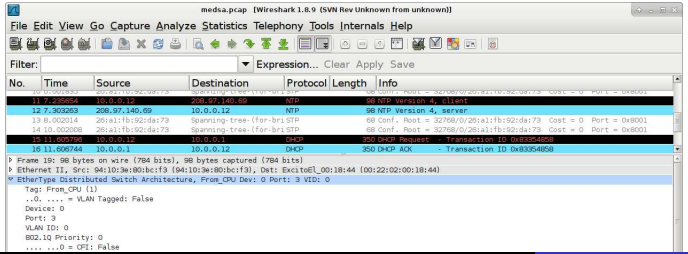
- Switch ports are Linux network interfaces
- Standard Linux tools used to configure interfaces, ip(8), ifconfig(8)
- Linux bridge concept used for bridging interfaces, ip(8), bridge(8), brtctl(8)
- Linux team/bonding concept for port trunks,
- The switch just accelerates what Linux can already do

DSA has been doing this since 2008!

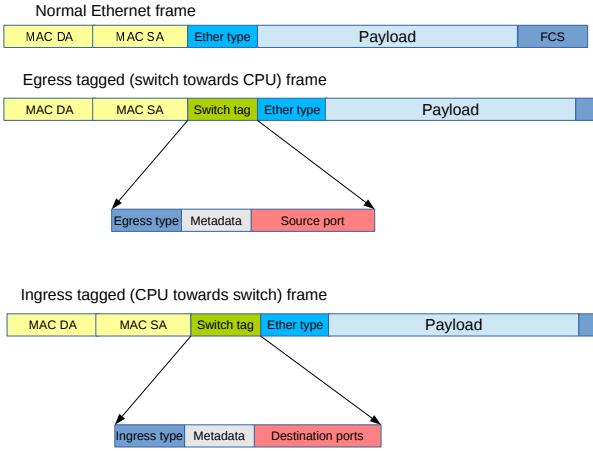
The Data Plane

DSA provides the data plane

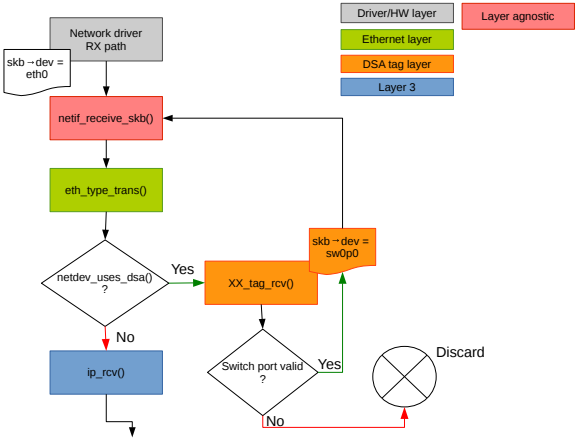
- Linux slave Interface for each port
- Tagging protocols, to direct frames from the SoC to a specific port
  - Taggers for Marvell DSA & EDSA, Broadcom and Qualcom, plus generic trailer
- TX: Slave interface -> tagger -> master interface -> Switch
- RX: master interface netif_receive_skb() -> tag parser, slave interface selection -> netif_receive_skb



Ethernet frame processing



Network stack flow



The Control Plane

About SWITCHDEV:

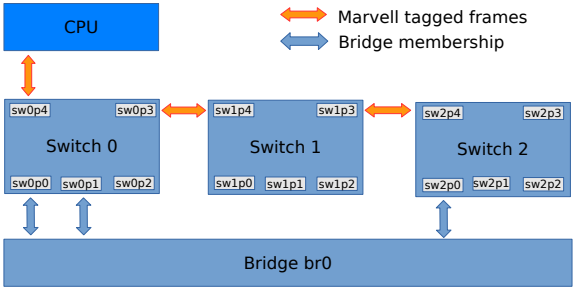
- Stateless framework in the kernel under net/switchdev/
- Provides the control knobs within the networking stack to push offloads towards specialized devices (switchdev_ops)
- Provides an abstract model of objects (VLANs, FDBs, MDBs,) to be pushed to these devices (switchdev_obj)
- Is not a device driver model: no strict definition of what a switch device is
- Only operates at the network device (net_device) layer

The Control Plane

DSA vs. SWITCHDEV:

- Stateful framework under net/dsa/
- Collection of vendor-specific switch tags: Marvell, Broadcom, Qualcomm, Mediatek
- Provides an abstracted model of a switch: dsa_switch and a collection of switches: dsa_switch_tree
- Binds network devices (netdev_ops), ethtool (ethtool_ops) and switch drivers together
- Well defined device (Device Tree) and driver model (dsa_switch_ops)
- Implements switchdev_ops for supported offloads: bridge, VLAN, FDB, MDB

Cross-chip configuration



Cross-chip configuration

Current behavior:

- Interconnected switch chips create a switch fabric
- DSA drivers manage single chip (struct dsa_switch)
- DSA links configured to pass frames to any port

Problem?

- br0 bridging? Switches can potentially leak cross-chip frames!
- br0 VLAN 42? Switch 1 won't pass traffic!

Cross-chip configuration

Solutions?

- Cross-chip bridging (DONE):
  - DSA core notifies drivers about fabric bridge events (crosschip_bridge_{join,leave} ops)
  - mv88e6xxx configure *Cross-chip Port Based VLAN Table* (PVT, Marvell specific)
- Cross-chip VLAN (WIP):
  - DSA core notifies drivers about switchdev port objects (VLAN, FDB, MDB) so that drivers allow traffic to pass

Hopefully coming soon

Multiple CPU ports

- Often seen in Wi-Fi devices to double SoC <-> Switch Bandwidth
- Often one CPU port statically mapped to “WAN” port
- Depends on chipset features, we can do better, load balancing

IGMP snooping on bridges
Better distributed switch support for Marvell devices
Better support for Fiber interfaces
Mediatek driver merged
Microchip drivers merged

Maybe Later???

Team/bonding?
TCAM support for offloading part of the firewall?
Qualcomm Hardware NAT?
More Vendor supported development?
Metering, broadcast storm suppression, QoS (priorities, maps) offloads again

Questions

Questions???