CUMULUS

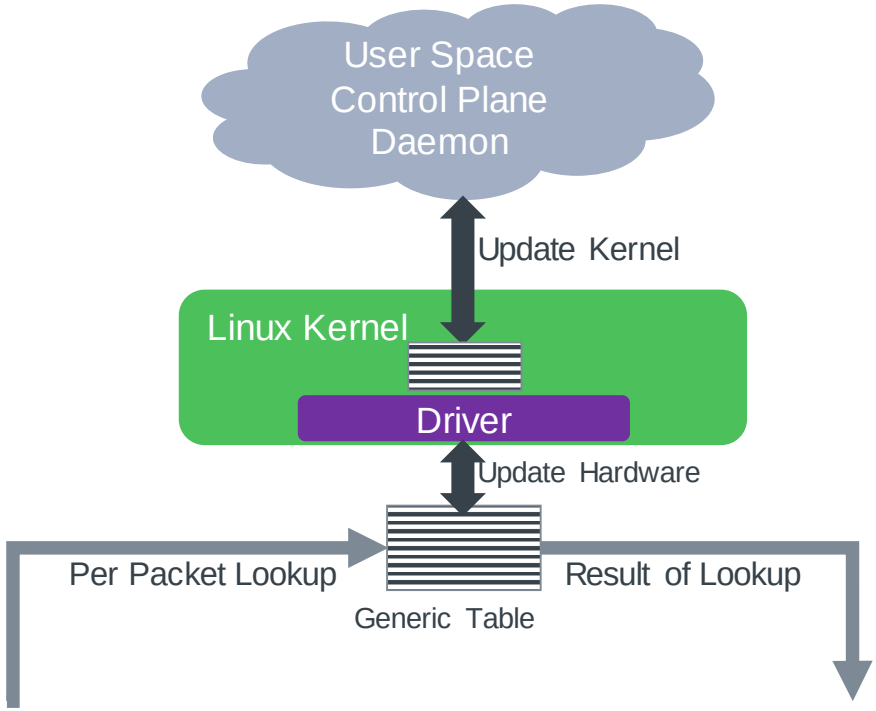
Resource Management in Offloaded Switches

Netdev 2.2 Seoul, Koreal

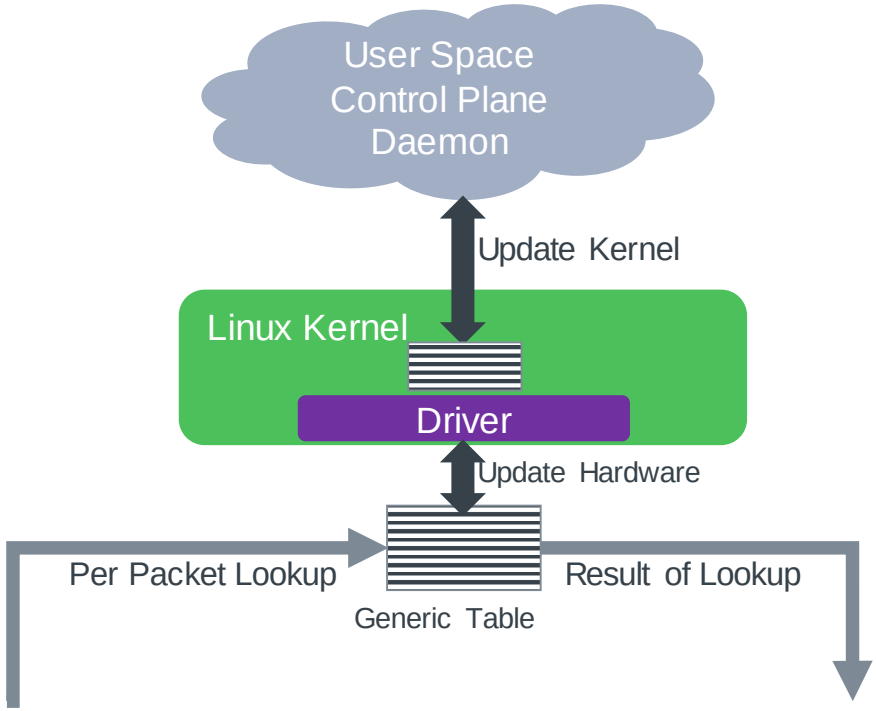
Nov, 2017

Andy Roulin, Shrijeet Mukherjee, David Ahern, Roopa Prabhu | Cumulus Networks

Some taxonomy



Steps which can fail



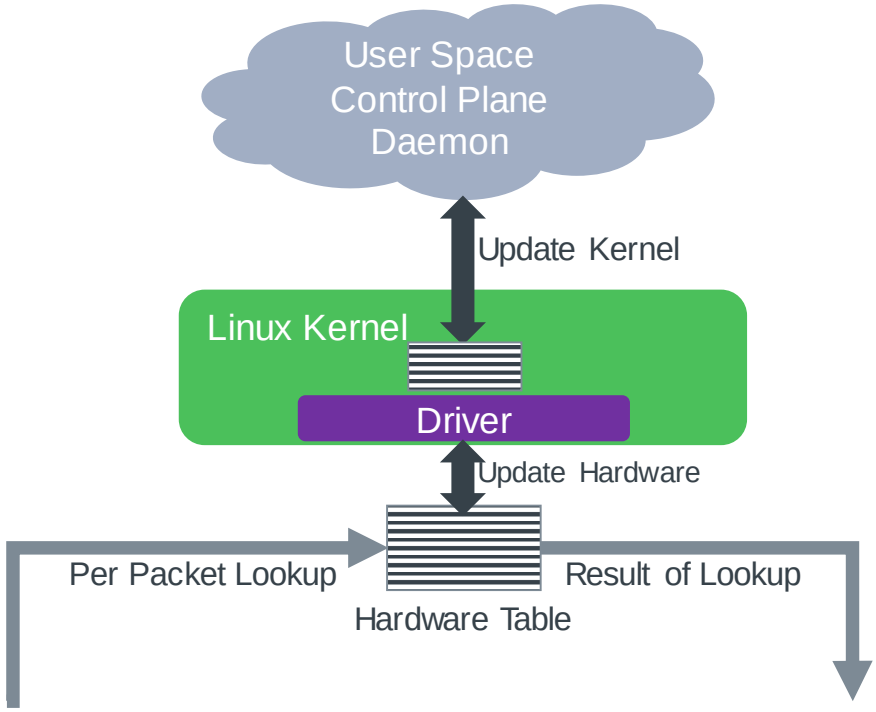
Updates from Control Plane to the Kernel

- Typically synchronous (netlink)
- Batching can cause effects
- Userspace handles it today

Updates from Kernel to hardware

- Capacity mismatches
- Rate mismatches
- Type mismatches

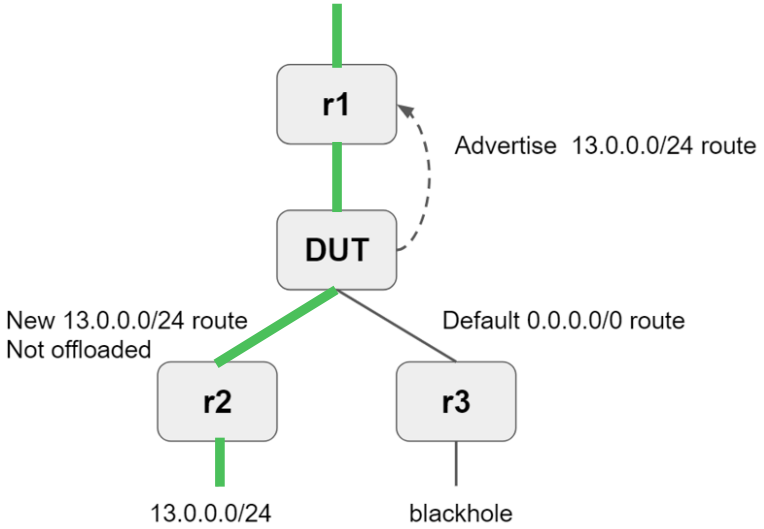
Types of Failures



1. Silent Failure
   - Kernel Table updated, Hardware table unchanged
   - Control plane oblivious
2. Advertised Failures
   - Failure propogates from Hardware to Kernel to Control plane
   - Control Plane needs special handling of asynchronous Failures

Synchronous Failure model not considered, covered later

An actual example – 4 routers linked by BGP



r1, r2 and r3 are routers

DUT connects them to each other

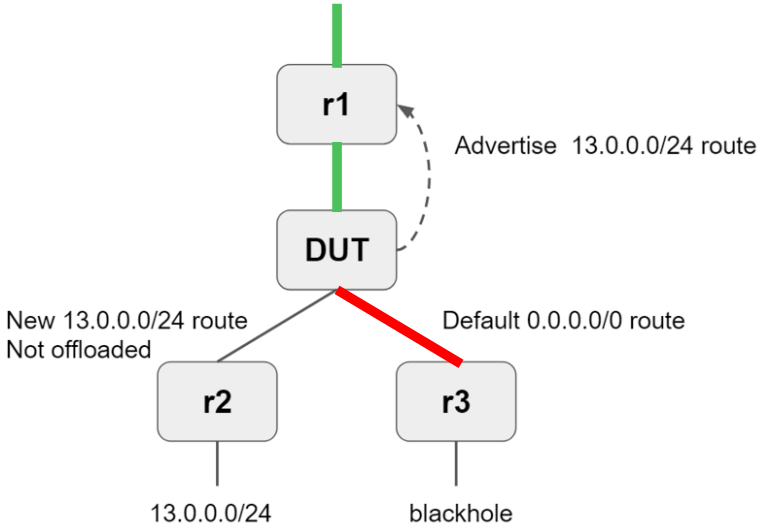
r3 is DUT's default route

- 13.0.0.0/24 advertised from r2
- Host 13.0.0.1 lives behind r2

Traffic from r1 is being sent towards r2 since

- DUT has learnt about 13.0.0.0/24 from r2
- DUT has advertised a path to 13.0.0.0/24 to r1

The Havoc the mis-programming creates



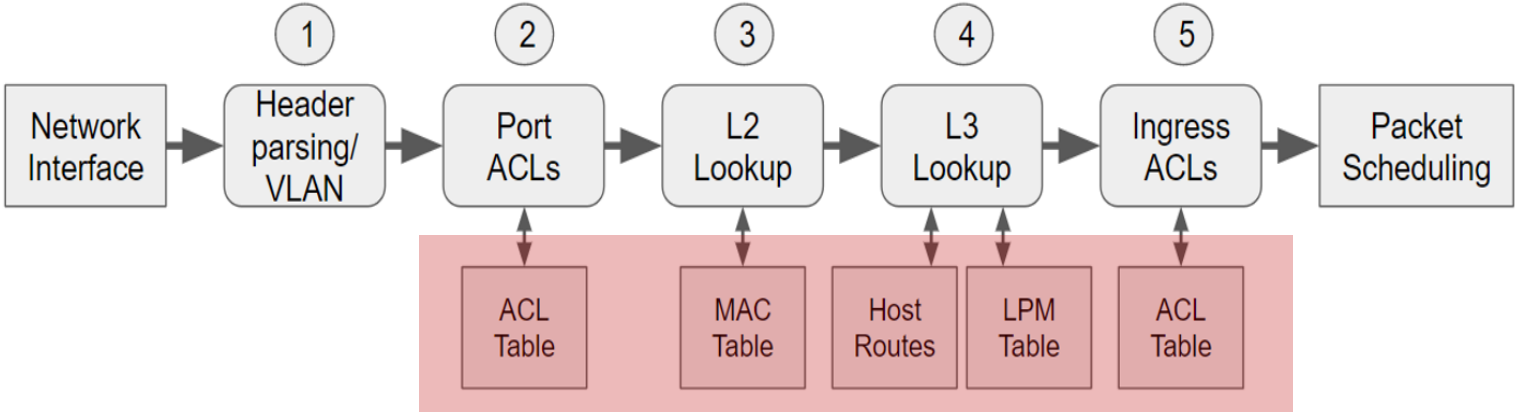
Now consider

- DUT accepted the route advertisement from r2
- DUT routing suite informed r1 of path to r2

HW however ran out of resources and failed the table update

- r1 has told all it's neighbors to send 13.0.0.0/24 traffic towards it as it can reach DUT
- DUT blackholes silently
- Happiness and Joy is felt all around

So how bad can the problem be



What needs to be offloaded in this model*

ACL Tables : Ingress and Egress

Mac Tables : L2 bridging data, vlans, vni's

Neighbor Table : directly connected hosts

LPM : Longest prefix match data for routing lookups

Data Center, high scale devices typically use logical tcams which are flexible and have dynamic behavior

Low scale home devices typically use real tcams and sram and have relatively predictable behavior

*This is not what all switching silicon implements, but most model functional blocks shown here

State of resource mgmt

ACL Tables

- netfilter offload has no clean structured path
- Implemented via ndo_setup_tc, which fails silently
- For switches this means
  - Punt all packets to CPU (impractical)
  - Only control plane packets have ACL in place (security hole)

Mac Tables

- Failure to install/fit an FDB entry will result in flooding
  - While sub-optimal typically functionality is not completely lost
  - In a large network, this is an unacceptable solution

State of resource mgmt (contd)

Neighbor Table

- Typically not subjected to oversubscription
  Bounded by total interface count in the system

L3 Configuration

- Configuration notifiers needs resources e.g. VRF creation, VxLAN device creation
- FIB add/delete notifiers need resources but are filled asynchronously
- Currently all Failures are silent
  Fallback to software not practical for data center class devices
  Failure results in network wide and hard to pin issues

Tables, Tables everywhere, how do we protect them all

This problem is not restricted to switching hardware

- Nics are exposed too

What is needed is

- Consistent offload failure error path to the user or protocol daemon;
- Signaling from kernel to userspace of resource utilization and capacity
- A resource manager model or resource management algorithm for drivers.

  Break things into user selectable profiles

  Allow shared memory to be used efficiently

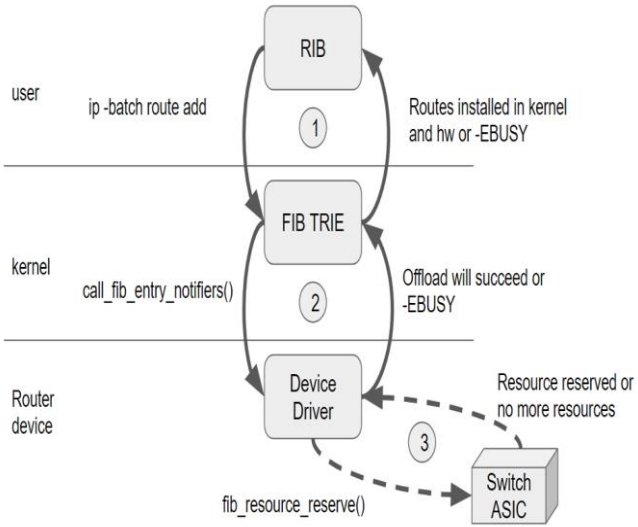
  More flexibility than the strict partitioning

Simple solutions

Driver profiles

- Dedicated resources can be accounted for easily
  Anything that is in a fixed table, TCAM implementations
- Try/Abort for complex resource allocations
  Complex calculations (e.g. ipv4 versus Ipv6/64 versus ipv6/128)
  Algorithm based TCAM implementations can fail at less than capacity
  Userspace protocol engines maybe able to recover over time

The experiment test bench



3 loops that matter

1. ip route add and it's return

   Standard mechanism here

2. The kernel using notifiers to the driver

   We added return value significance here

3. The vendor driver updating the real hardware

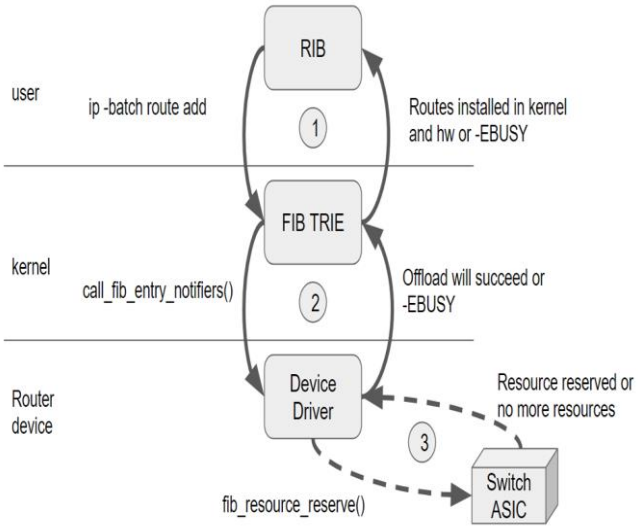
   Simulated H/W query as a usleep

   Put in synchronous check of query result in fib notifier path

   Optionally : async workqueue emulating calling H/W query and refilling the resources available

Actual code changes are here :

Various models explored in this paper



Lockstep

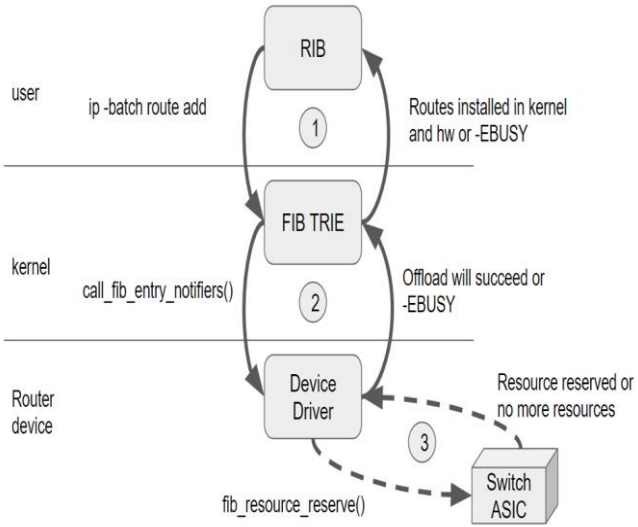
- Explicit capacity check
- Simple and naïve implementation
- All loops are executed in series

Synchronous prefetch

- Capacity within skid and hides latency
- Above solution + prefetch of a batch of resources
- Latency amortized over batch size

Actual code changes are here :

Various models explored in this paper

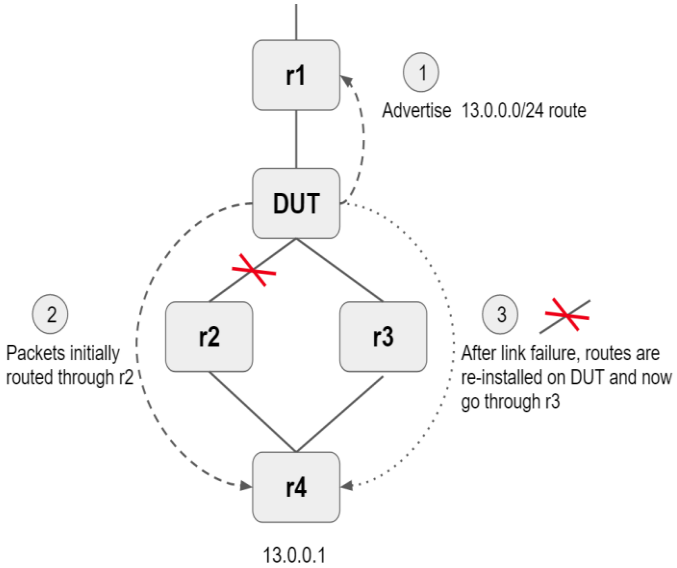


Credit based prefetch

- Hides latency, checks capacity and rate
- Asynchronous update of availability
- Matches capacity and rate
- Needs atomic add from async thread

Actual code changes are here :

One solution that works everywhere .. ish



Measurements for 10000 routes being updated from FRR to hardware

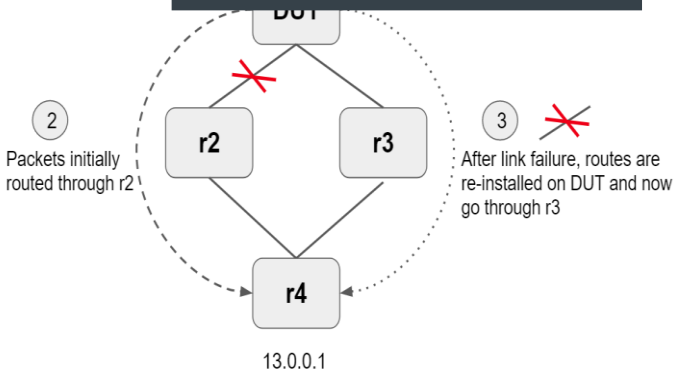
Method	Users Latency	HW Latency	Query Latency	Data Outage
Lock Step	600ms	60us	35us	1.10s
Credit based	480ms	60us	35us	0.81s
Lock Step	600ms	60us	350us	3.98s
Credit based	480ms	60us	350us	0.84s

Remember each trip to hardware takes 50ish usecs, which is 500ms unless they are batched

One solution that works everywhere .. ish

This is really 55us * 10000 and some batching help

Increased H/W latency has no implication



Measurements for 10000 ro … **o hardware**

Method	Users Latenc			Data Outage
L… Step	600ms	60us		1.10s
Credit based	480ms	60us		0.81s
Lock Step	600ms	60us	3…us	3.98s
Credit based	480ms	60us	350us	0.84s

Remember each trip to hardware takes 50ish usecs, which is 500ms unless they are batched

Conclusions

Resource management is non-negotiable for user experience

A credit queue based scheme will work well

- It also hides h/w and transaction latency well
- Fits in with minimal changes into our current system
- Should have no foot print in pure s/w path
- If same principle was applied to the write path

  The total transaction latency can be reduced dramatically

Thank you!

Visit us at cumulusnetworks.com or follow us @cumulusnetworks