Miscellany

I’m not good at grand unifying themes…

- RX batching (“skbundle”)
- Merging OVS megaflows
- Loose GRO
- ARFS flapping issue
- Inevitable eBPF-verifier argument

Edward Cree

Netconf ‘18, Boston MA

Batching/bulking design

- RFC patches[1] back in April 2016
  - Were not well received
  - But now Spectre may revive interest
- Leverages existing network stack
  - No early demux, no “crazy hashes across 57 headers”, no second implementation to get out of sync with the main one
- Unit of work is the SKB list

[1] http://lists.openwall.net/netdev/2016/04/19/89

Listification algorithm

- Simple case: “if (skb_is_x(skb)) { do_x(skb); } else { do_not_x(skb); }”
  - Example is PFMEMALLOC
- Have a “current list”, and track whether it’s a list of x-es or not-x-es.
- When next skb in source list doesn’t match current list, dispatch current list
  - “if (list_is_x) { do_x_list(list); } else { do_not_x_list(list); }”
- Then start a new list with next skb

Listification algorithm

- More complex cases – packet_type lookup sent SKBs singly to taps (e.g. tcpdump), list to last recipient only
  - But could change this, at cost of more complex code
- Indirect calls add a →handle_list() method, if it’s absent then call →handle() in a loop.
  - “sameness” for list is if the struct we’re getting these methods from is the same.

Listification principle

- Absolute minimum of new “network” code
  - Only “list manglement” code in new functions
- Done by factoring out all those “skb_is_x()” and “do_x_skb()” routines
- NOT a parallel implementation of the datapath, just a different way of hooking up the bits of the existing datapath
- Split lists when you find you need to, don’t try to do it in advance (like GRO does)

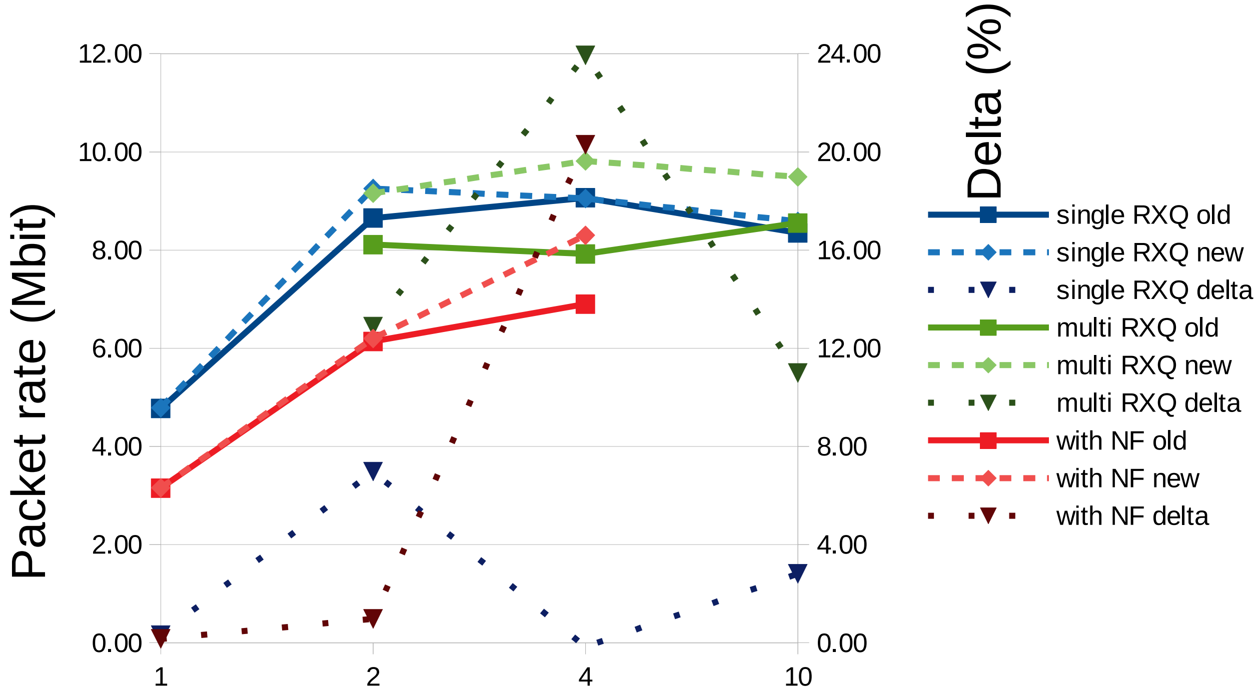
Listification results

- Original results are well out of date now (pre-Spectre, and kernel has moved on in 2 years)
- Saw 25% improvement in packet rate
  - 1-byte UDP packets
  - 10 TX streams, 1 RX queue
  - No firewall/iptables/tc/netfilter/etc...
- Only went as far as IP layer
- Design inherently allows gradual/creeping listification of the stack, no flag day needed

Listification results

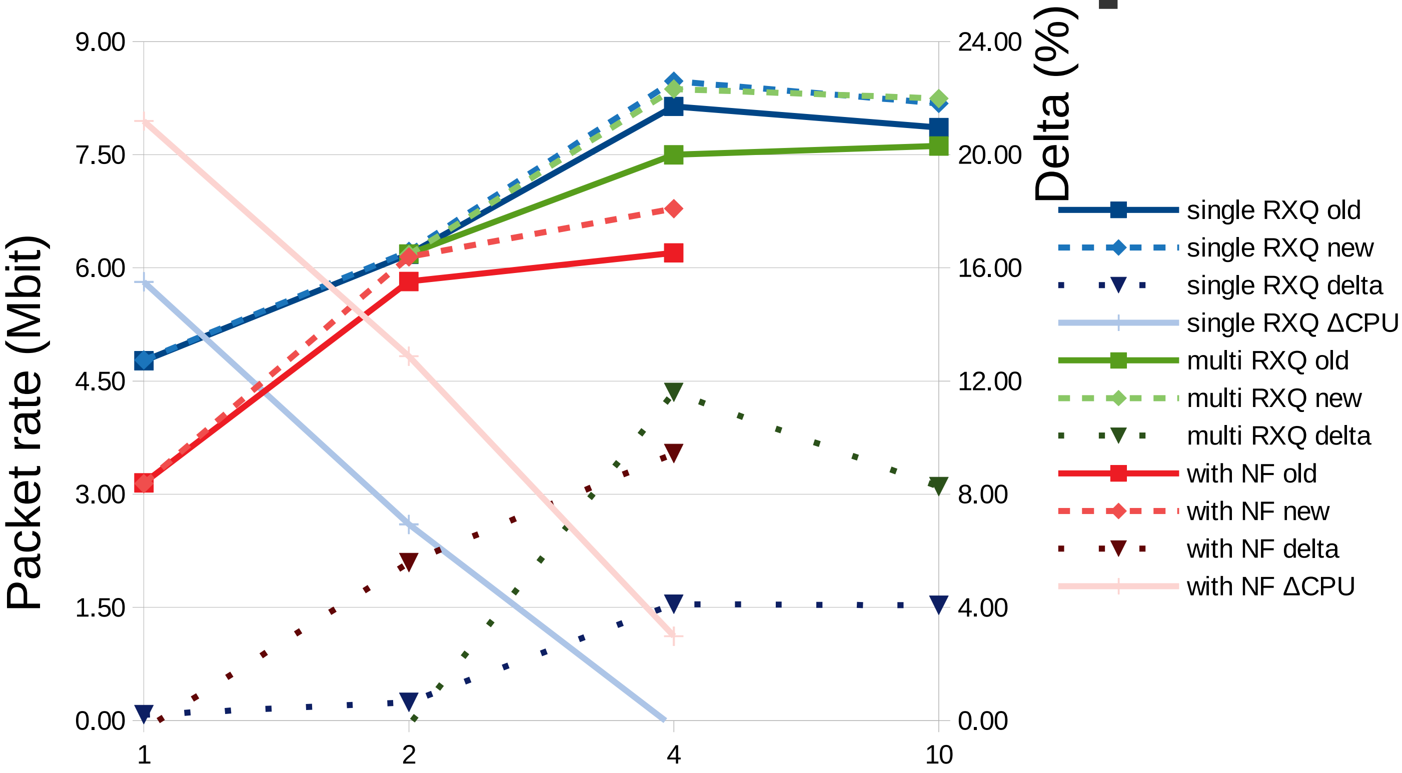
- Have rebased onto net-next, re-tested (**without** a retpoline-capable gcc!)
- Testing is again UDP streams to a single cpu
  - NetPerf, IRQs on one CPU, app on another
  - Either with just one HW RXQ, or several all pinned to the same CPU
  - HW is Solarflare SFN8522 10GbE adaptor, dellr710 server
- Also ran tests with a single iptables rule (port drop on a range not matching the traffic)

Listification results



- Very variable, from negligible change up to more than 20% increase.
- 1-stream cases TX limited, ~20% decrease in RX CPU usage.

Listification results - retpoline



- For few streams, up to 20% less CPU usage.
- Many streams, throughput increases by a few percent. It’s not huge.
  - Perhaps partly because only one indirect call (ip_rcv) listified so far

Listification plans

- Some further cleanup of the patches
- Perf measures with “real-world” firewall/netfilter/etc were demanded before
  - So can someone give me a real-world ruleset to test with?
  - Unlike some people here I can’t just ask the datacenter sysadmins for a copy
- Measure forwarding perf (e.g. RFC2544)
  - maybe even with lists all the way to TX...
  - just `skb->xmit_more`, not a new ndo

Listification plans

- Listified XDP-generic (because, indirect call)
  - would need JITs to produce list-unwrap loop in prologue/epilogue
  - User-supplied eBPF program doesn’t need to know
- Push further up the stack
  - e.g. TCP/UDP can go all the way to socket enqueue
  - Might see diminishing returns, but also might see the opposite, just don’t know.

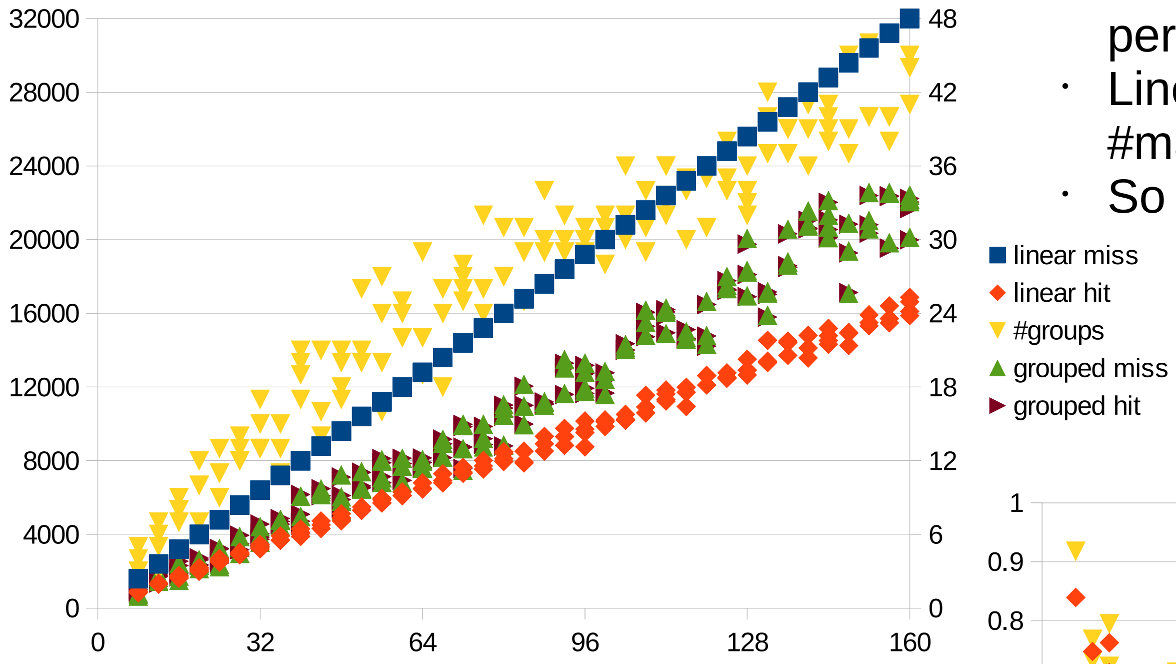
Megaflow merging

- Idea to improve performance of OVS datapath / other flower-like masked matches
- Find megaflows (masksets) with nearly the same masks
- Hierarchical lookup, mask of group is intersection of member masks
  - Group’s hashtable tells which members might match
- Tested a random model (likely worse than real-world masksets).

Megaflow merging - model

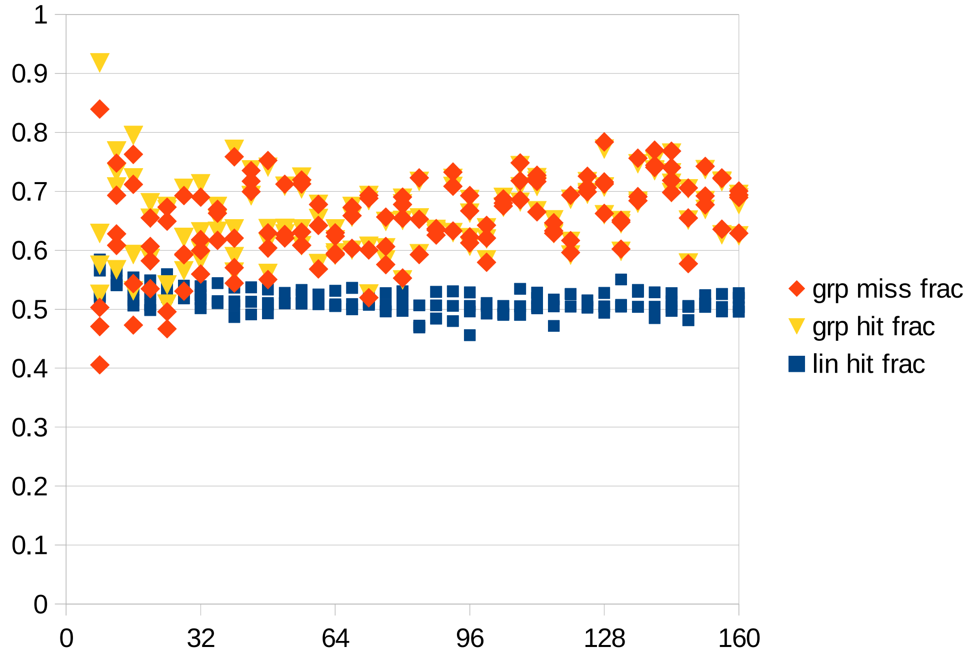
- Simple abstracted model, written in Python
- Has 72 bits of key, masksets are random bitmasks
- Each megaflow has rand(1, 100) filters
- Code available on request
  - But it’s not pretty

Megaflow merging - results



- Using number of hash lookups as perf metric
- Linear miss is predictable – always #masksets * #tests
- So view others as fraction of that

- Predictable performance (same in hit & miss cases)
- Faster than linear miss
- But slower than linear hit
- Trade-off worth it?
- Want real-world masksets to do more relevant/realistic tests.



Megaflow merging - plans

- Keep experimenting with merge heuristics
  - Currently greedy algorithm driven by Hamming distances
  - Detects “useless” groups and repartitions
  - Quite slow, even by Python standards
- Take a leaf out of bpfilter’s book
  - Finding good groups is intensive calculation – do it in userspace

Loose GRO

- Wacky untested idea
- If sysadmin knows they’re not doing forwarding, don’t need the “GRO guarantee”
- So can coalesce more packets
- This is the only reason people still want the “soft LRO” in the out-of-tree sfc driver
  - Which I’d love to get rid of
  - Smaller diff between in-tree and out-of-tree makes my life a lot easier ;-)

ARFS flap

- Caused by misconfigured interrupt affinities
- cpu→rxq map is many-to-many
  - So current “rxq == skb rxq” is too strict
  - Instead use the reverse-reverse-map
  - Check if cpu in irqmask of skb rxq
- Unfinished project, urgency went away after we fixed sfc’s irq affinity hints
  - But user could still create the bad config
- Have patch but haven’t proved it fixes problem.

Backup

If we have time for an argument

eBPF Verifier

These are just opinions; I’m open to being convinced to change them…

- Can’t put off the showdown any longer
- What’s good for compilers in userspace is not always good for verifier in the kernel
- Data structures should be implicit unless there’s a reason to make them textbook
- Hence register-parent chains rather than dom trees, subprog/bb indices and callee/r bitmasks in aux data rather than pointer tree replicating control-flow graph
- Avoiding full walk maybe mathematically impossible, learn to like it