What's happened to the world of networking hardware offloads?

Jesse Brandeburg
Anjali Singhai Jain

Agenda

Introductions

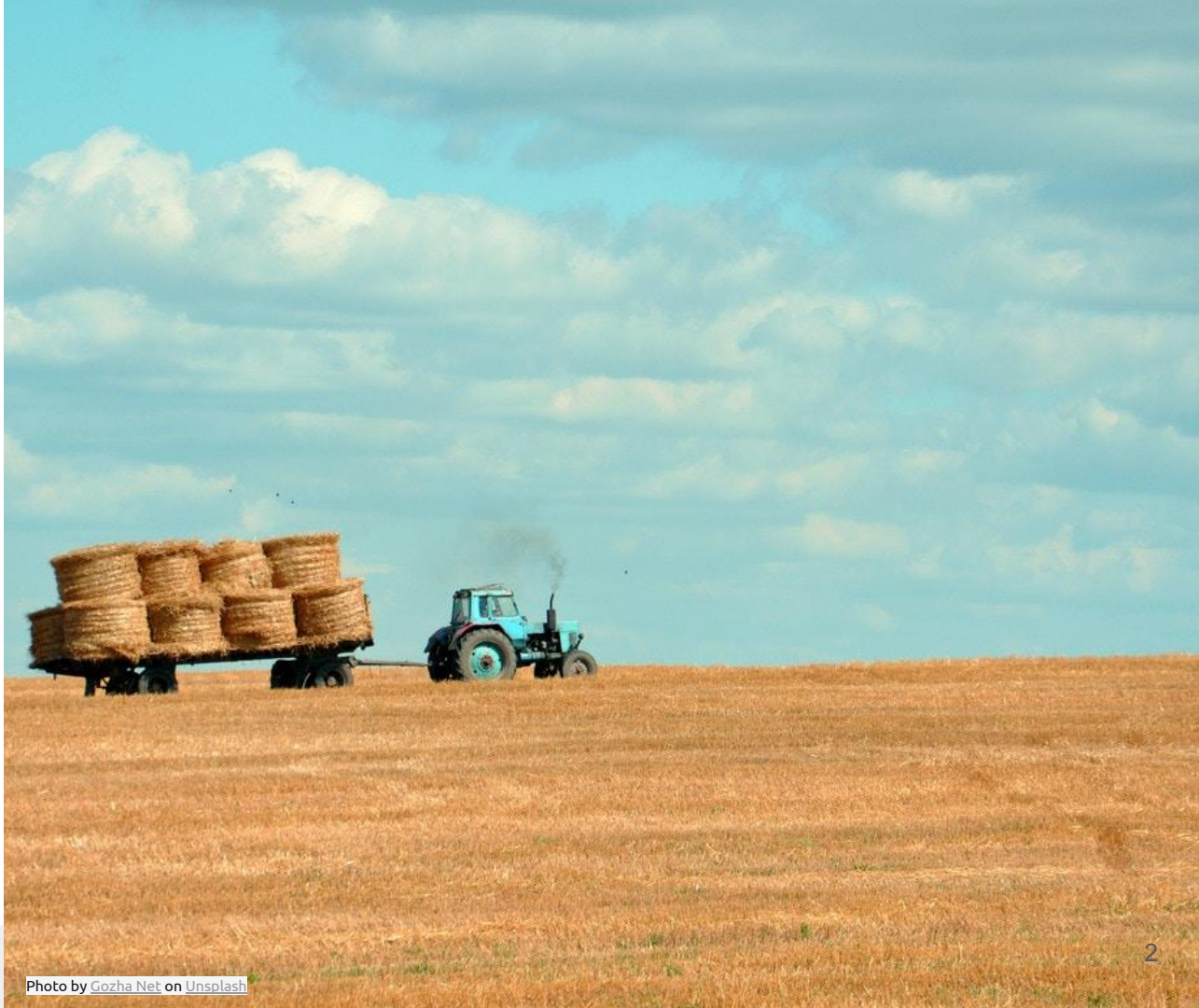
A Brief History of Offloads

Hardware

Offloads

Future Look

Proposals



Introductions

Anjali Singhai Jain

Jesse Brandeburg

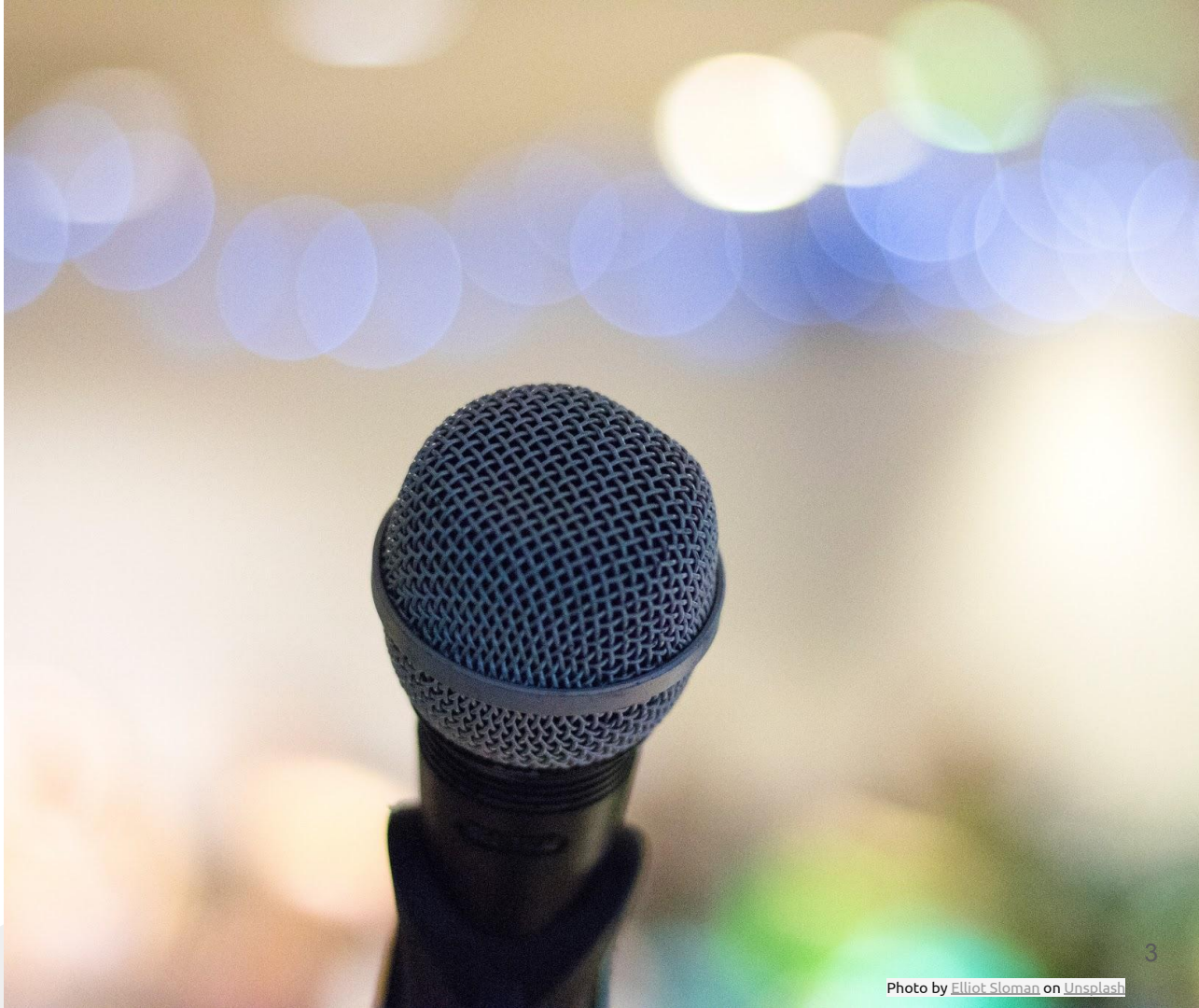
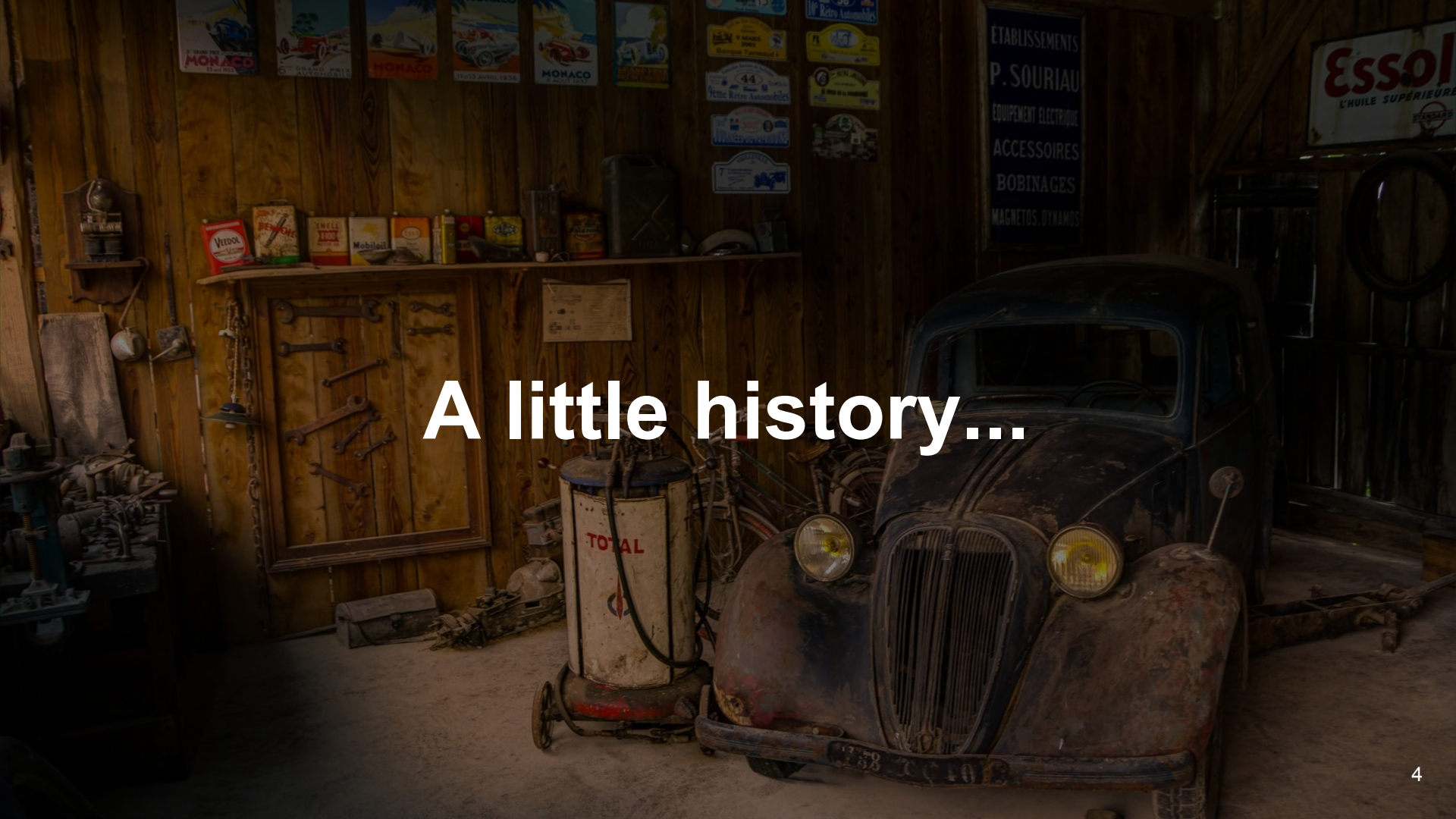


Photo by Elliot Sloman on Unsplash



A little history...

A Brief History of Offloads

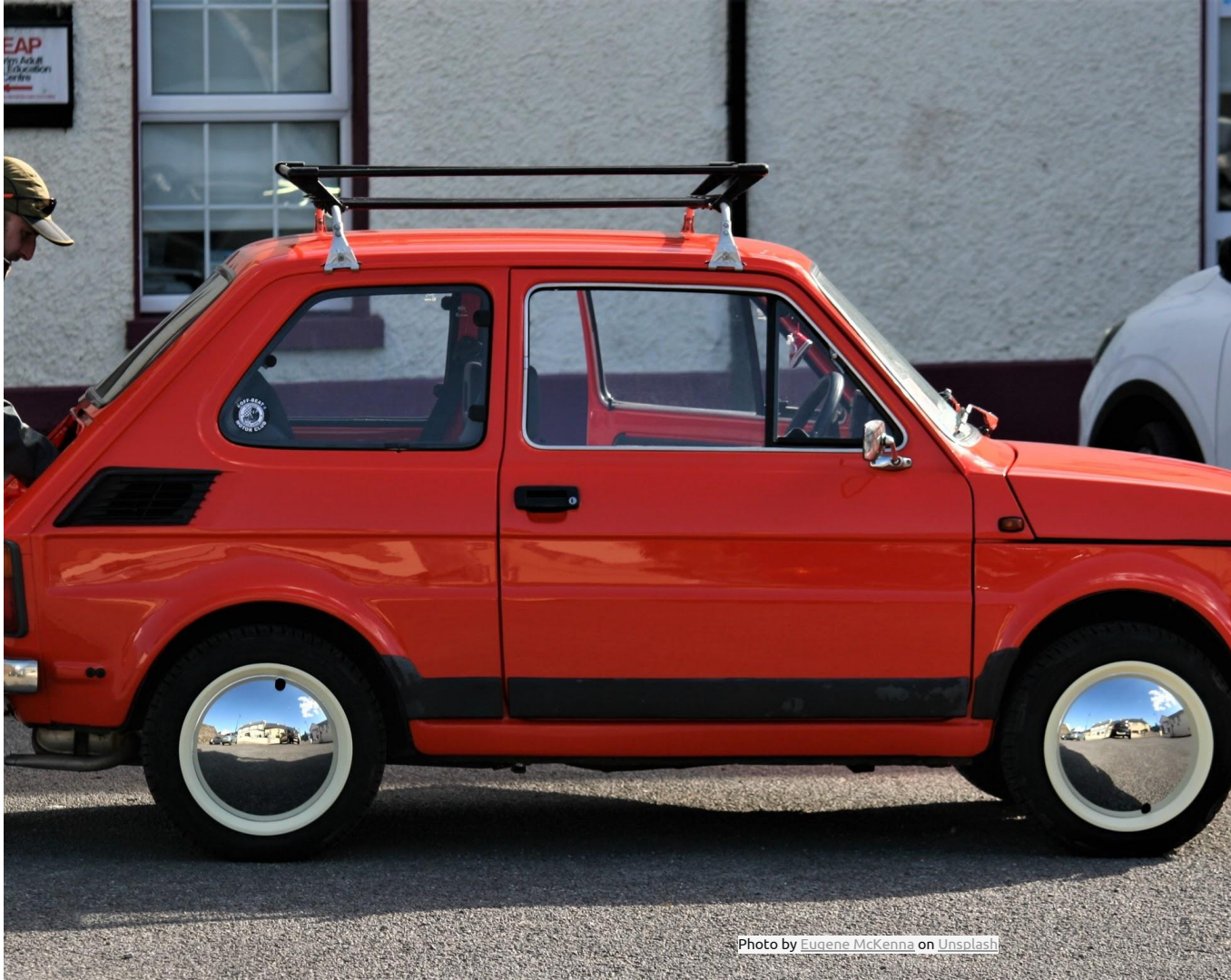
Why offload?

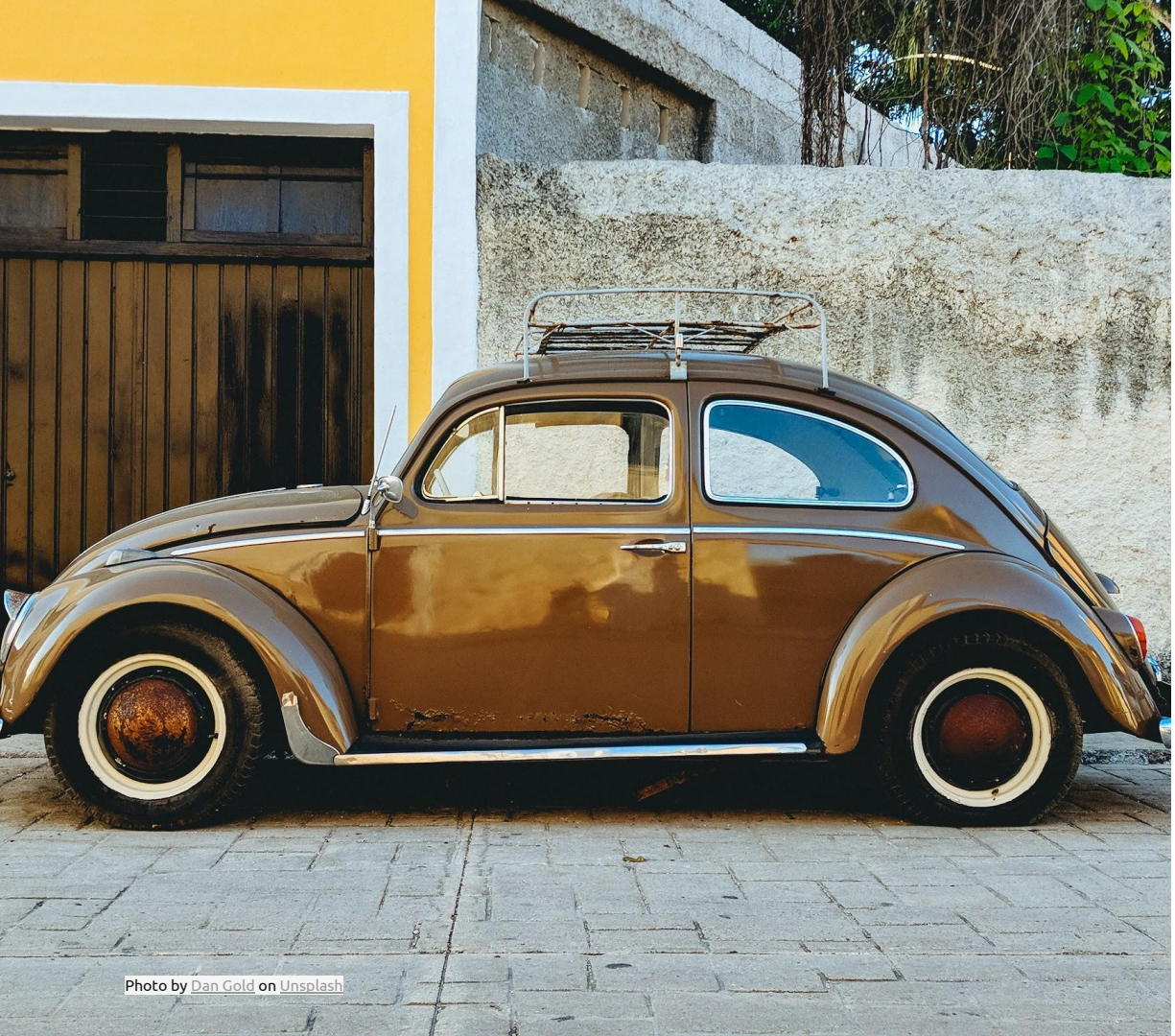
It all began with a small set of offloads

SG - scatter gather

IP CSUM - can insert ip checksum

HW CSUM - can checksum (most) everything





History (cont.)

More offloads!

802.1q VLAN insert and delete

Transmit Segmentation Offload

Heavyweight stack changes were necessary

Even more history

Quite a few more offloads added

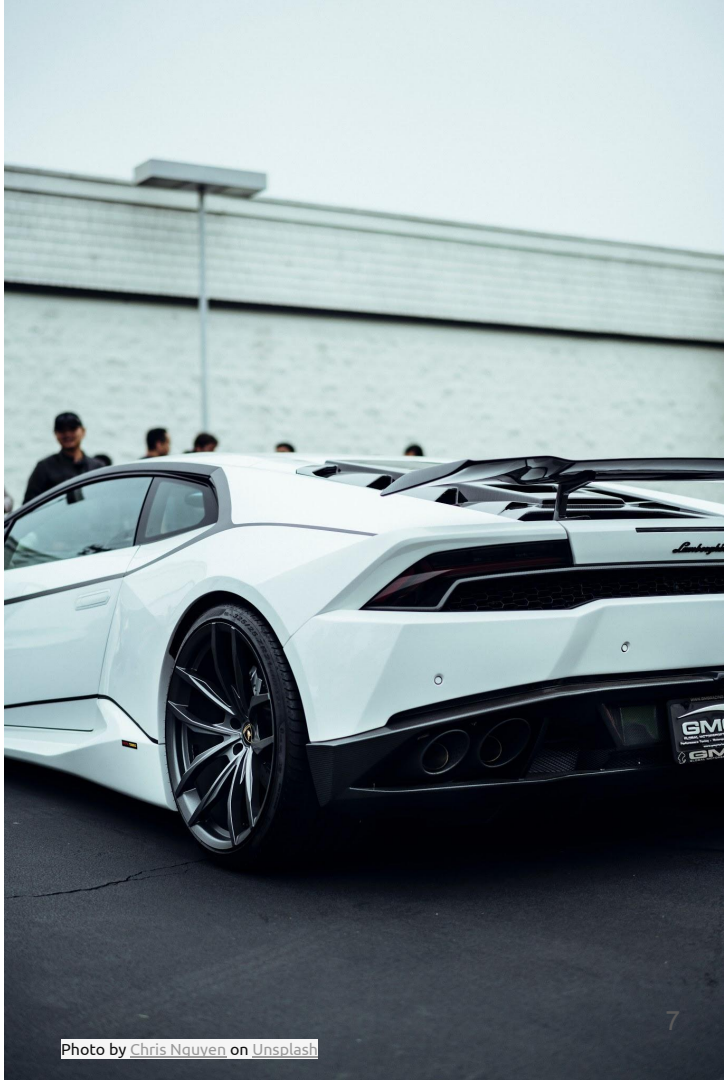
Most are transmit offloads

Less receive offloads, but it’s a growth area

Implementations are getting quite a bit more complex (more hardware)

No longer stateless only; moving logical flows into hardware, like eBPF

Also creating new paradigms, like tc-flower for vSwitch offload



Behemoth

We now have switches in the NIC

FPGAs, CPUs, RAM

Virtualization

Tunnel Offloads
- TSO, csum
- encap / decap / TEP in hardware

Flow Tracking
- Millions of rules / counters for flows

Lots of hardware or driver based controls possible

Interfaces don’t scale

Only small overlap between vendors

Photo by Phil Whitehouse on Flickr

ethtool --show-offloads

```
# ethtool -k ens2f0
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: on
        tx-checksum-ip-generic: off [fixed]
        tx-checksum-ipv6: on
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: on
scatter-gather: on
        tx-scatter-gather: on
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
        tx-tcp-ecn-segmentation: on
        tx-tcp-mangleid-segmentation: off
        tx-tcp6-segmentation: on
udp-fragmentation-offload: off
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: off [fixed]
rx-vlan-offload: on
tx-vlan-offload: on
ntuple-filters: off
receive-hashing: on
highdma: on
rx-vlan-filter: on [fixed]
vlan-challenged: off [fixed]
tx-lockless: off [fixed]
netns-local: off [fixed]
tx-gso-robust: off [fixed]
tx-fcoe-segmentation: off [fixed]
tx-gre-segmentation: on
tx-gre-csum-segmentation: on
tx-ipxip4-segmentation: off [fixed]
tx-ipxip6-segmentation: off [fixed]
tx-udp_tnl-segmentation: on
tx-udp_tnl-csum-segmentation: on
tx-gso-partial: on
tx-sctp-segmentation: off [fixed]
tx-esp-segmentation: off [fixed]
tx-udp-segmentation: off [fixed]
fcoe-mtu: off [fixed]
tx-nocache-copy: off
loopback: off [fixed]
rx-fcs: off [fixed]
rx-all: off [fixed]
tx-vlan-stag-hw-insert: off [fixed]
rx-vlan-stag-hw-parse: off [fixed]
rx-vlan-stag-filter: off [fixed]
l2-fwd-offload: off [fixed]
hw-tc-offload: off
esp-hw-offload: off [fixed]
esp-tx-csum-hw-offload: off [fixed]
rx-udp_tunnel-port-offload: on
tls-hw-tx-offload: off [fixed]
rx-gro-hw: off [fixed]
tls-hw-record: off [fixed]
```

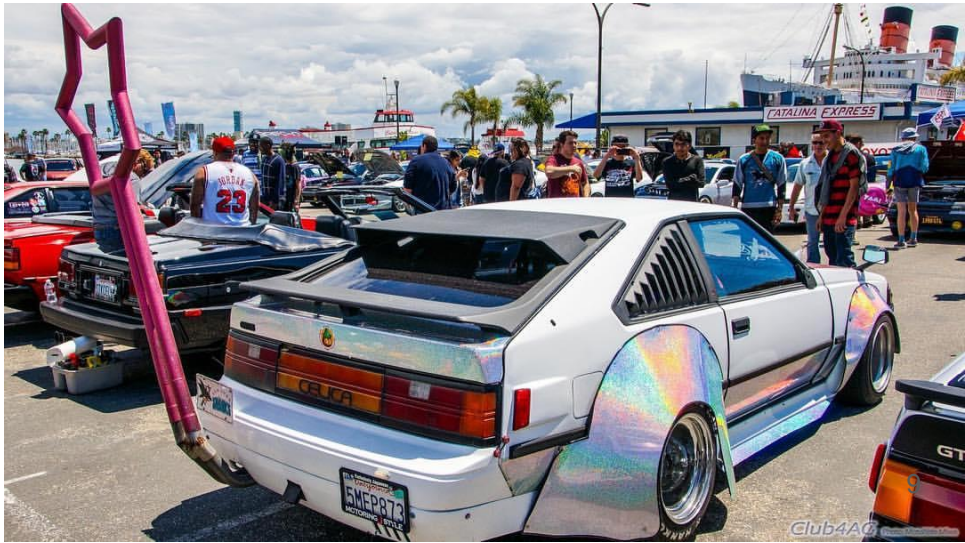


Photo by Moto "Club4AG" Miwa on Flickr

Option Overload

The granularity of offload advertisement is too coarse

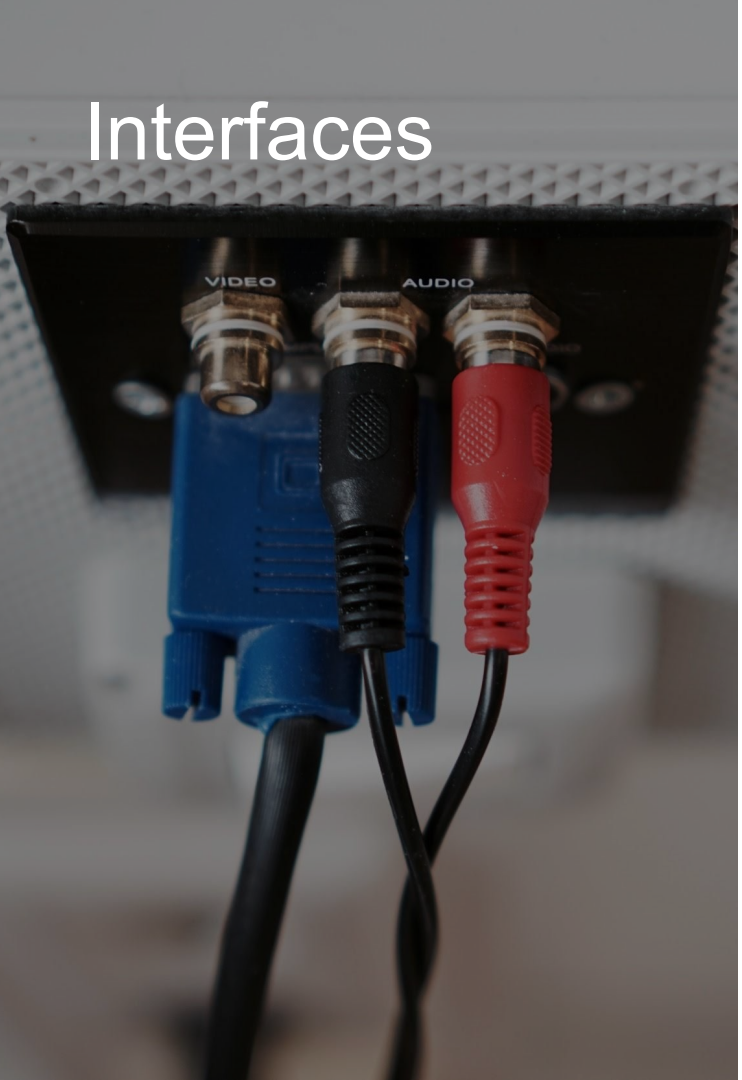
Perfect example:

ethtool --hw-tc-offload

Any match, any action? How many flow rules? Silent failures!

Huge vendor specific documentation to actually elaborate what that offload means

Interfaces



ethtool

- Queues
- Interrupt Rates
- RSS tables
- Ntuple rules (yuck)
- Hardware offloads
- External port properties (speed, duplex, etc)

devlink

- Configure devices
  - eswitch
  - New param option
- dpipe

iproute2

- tc
  - u32
  - flower
- ip
  - MTU, link speed, namespace, vf control, xdp, xfrm, etc

eBPF / XDP

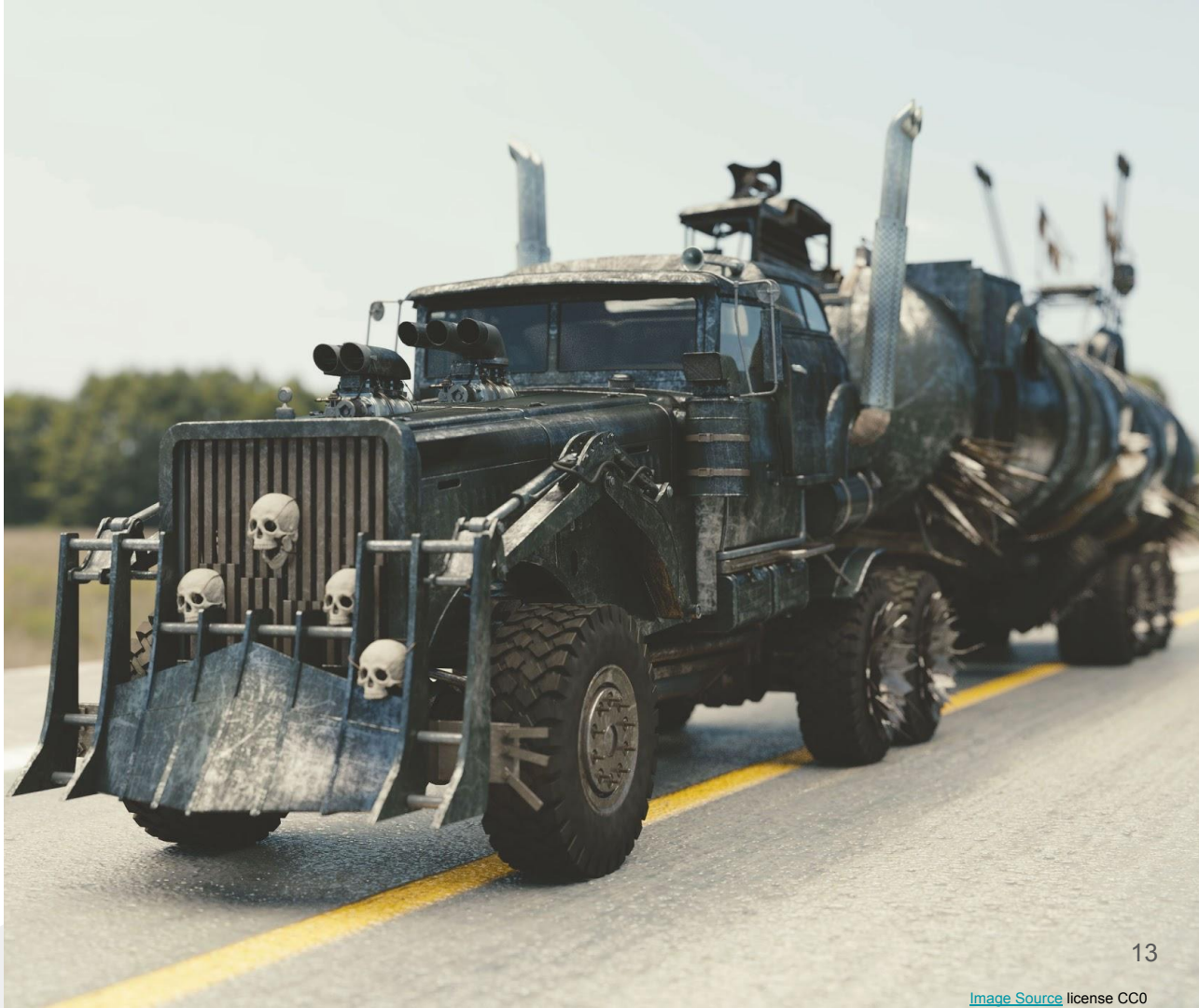
- No programmatic control of offloads
- See metadata presentation

The netdevs are not the external ports!

Problem

Organically grown solutions to many problems over many years

Each one works on it’s own, but are we building the design we want at the end?



The Challenge

The old models don’t fit well anymore

Come up with a kernel compatible method of allowing expression and use of complex network device features

Is devlink dev param the right direction?

The Idea

Generic offload expression

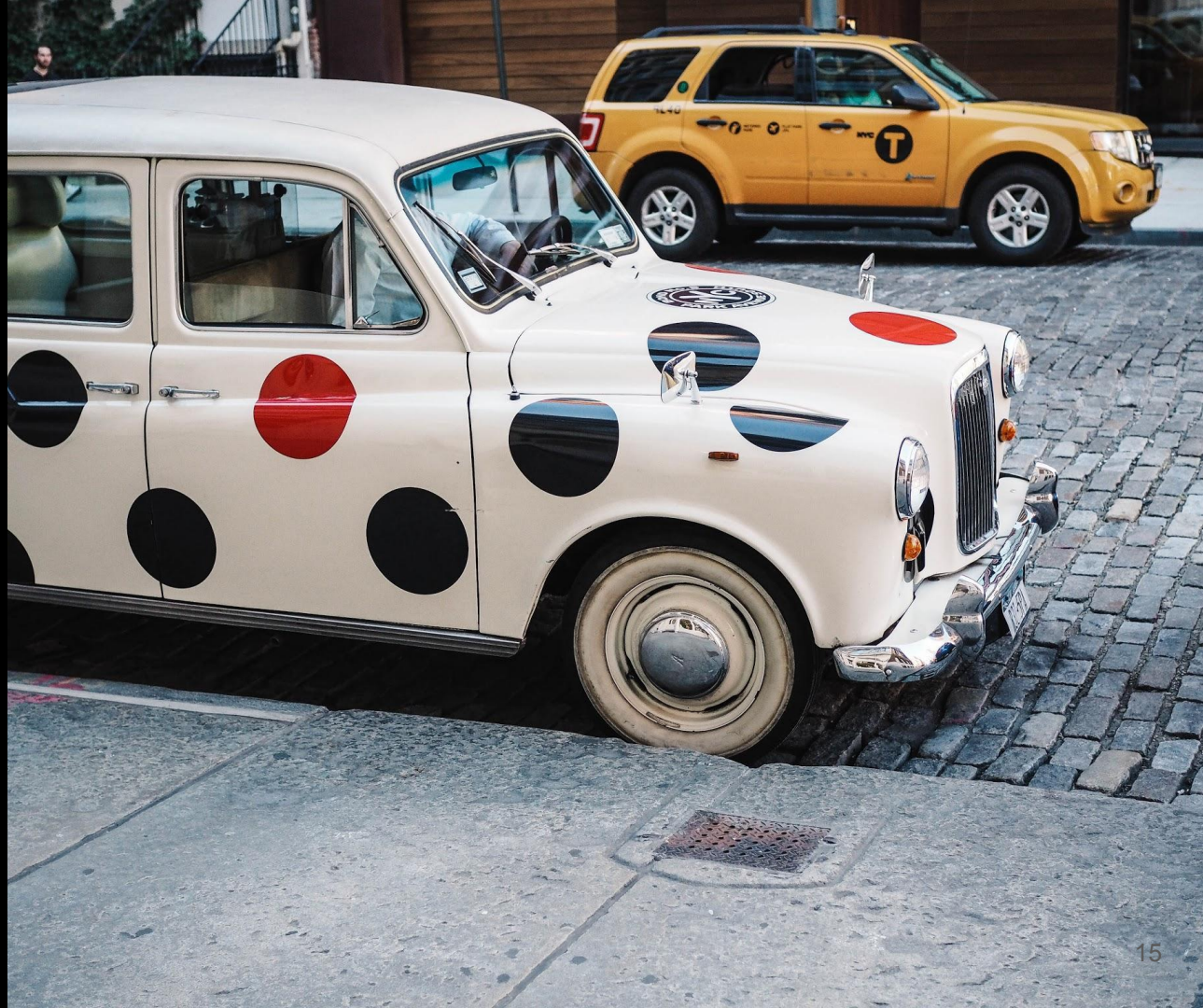
Name value pairs

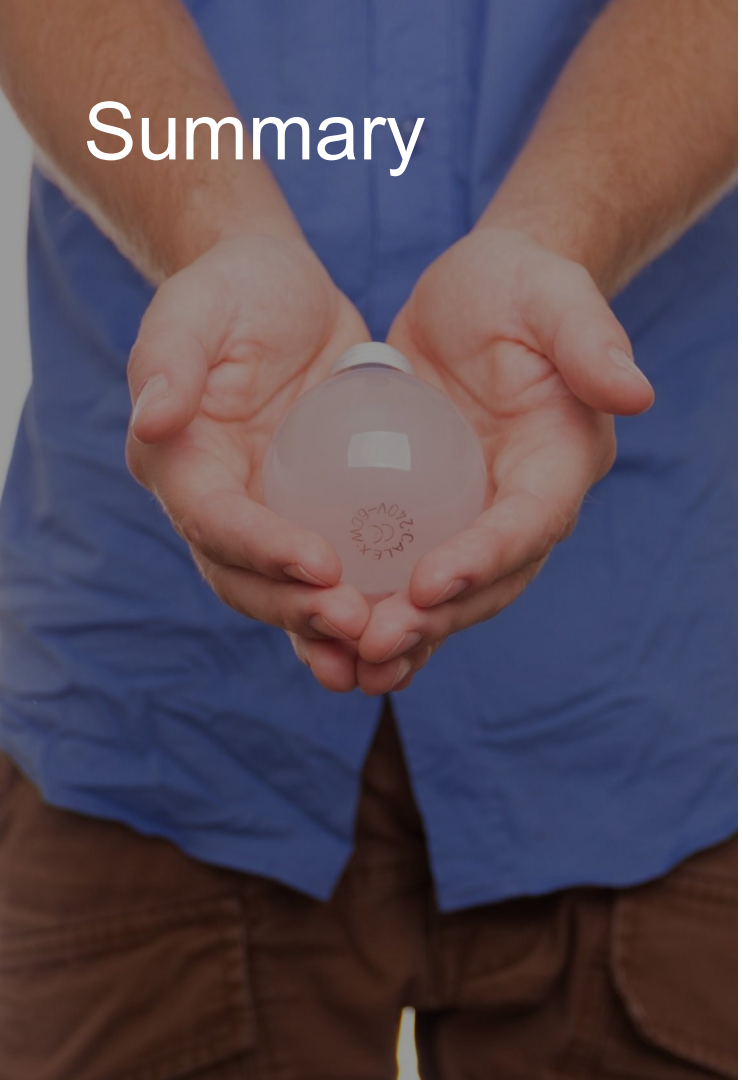
Common offload object

Need to communicate both common capability and unique elements

Need a user (lib?) and kernel implementation

Much like devlink dev param show/set





Summary

Offload infrastructure code is needed

Complicated hardware offloads

- Configuration is necessary
- Linux doesn’t have a good way to express limits or configuration
- Need granular control

Everything defaults to the lowest common denominator or worse

The community could benefit (and us too) from good ideas in this space and some help to implement a workable solution

Questions?