Google

Unsafe at Any Speed?

Self-Driving Networks without Self-Crashing Networks

Jeff Mogul
Google Network Infrastructure
24 August 2018

"Self-driving cars": a poor template for Self-Driving Networks

How to build a self-driving car:

- start with a late-model car
- add lots of sensors, computers, and AI software
- train and test it until it works safely

That worked out much sooner than most people expected

"Self-driving cars": a poor template for Self-Driving Networks

How to build a self-driving car:

- start with a late-model car
- add lots of sensors, computers, and AI software
- train and test it until it works safely

That worked out much sooner than most people expected

So why not build a self-driving network the same way?

- start with a modern network
- add lots of sensors, computers, and AI software
- train and test it until it works safely

"Self-driving cars": a poor template for Self-Driving Networks

How to build a self-driving car:

- start with a late-model car that any adult can drive safely, even with accidents
- add lots of sensors, computers, and AI software
- train and test it until it works safely

That worked out much sooner than most people expected

So why not build a self-driving network the same way?

- start with a modern network that any operator can run without frequent outages
- add lots of sensors, computers, and AI software
- train and test it until it works safely

"Self-driving cars": a poor template for Self-Driving Networks

How to build a self-driving car:

- start with a late-model car that any adult can drive safely, even with accidents
- add lots of sensors, computers, and AI software
- train and test it until it works safely

These exist

That worked out much sooner than most people expected

These do not exist!

So why not build a self-driving network the same way?

- start with a modern network ~~that any operator can run without frequent outages~~
- add lots of sensors, computers, and AI software
- train and test it until it works safely

Safety: A key challenge for self-driving networks

We need our networks to be "safe":

- i.e., they meet many kinds of SLOs (uptime, bandwidth, latency)

Today's networks require a *lot* of human effort to maintain SLOs

- and replacing humans with AI doesn't work *if the problem is unnecessarily hard*

This talk is about meeting that challenge, and what we can learn from the past

Alternate title: *Nobody would build a self-driving Corvair*

Alternate title: *Nobody would build a self-driving Corvair*

"The Chevrolet Corvair is a compact car manufactured by Chevrolet for model years 1960–1969. It was the only American-designed, mass-produced passenger car to use a rear-mounted, air-cooled engine." [Wikipedia]

Alternate title: *Nobody would build a self-driving Corvair*

"The Chevrolet Corvair is a compact car manufactured by Chevrolet for model years 1960–1969. It was the only American-designed, mass-produced passenger car to use a rear-mounted, air-cooled engine." [Wikipedia]

… and became notorious as the primary example of an inherently-unsafe car, as described by Ralph Nader in *Unsafe at Any Speed*

Alternate title: *Nobody would build a self-driving Corvair*

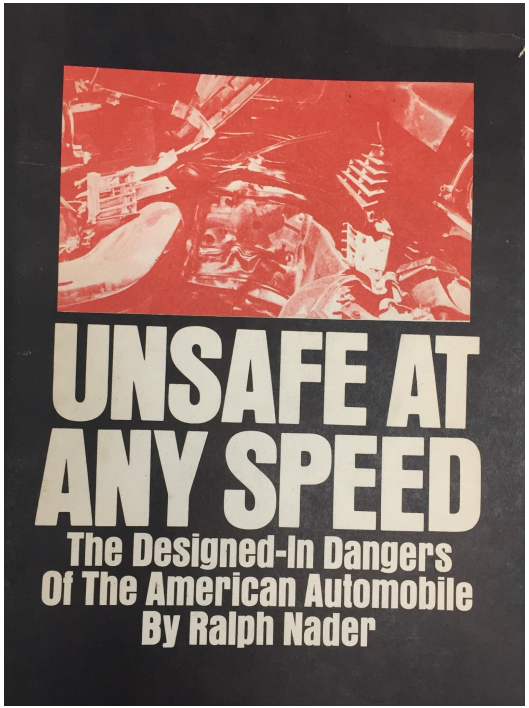
"The Chevrolet Corvair is a compact car manufactured by Chevrolet for model years 1960–1969. It was the only American-designed, mass-produced passenger car to use a rear-mounted, air-cooled engine." [Wikipedia]

… and became notorious as the primary example of an inherently-unsafe car, as described by Ralph Nader in *Unsafe at Any Speed*

… although subsequent studies suggested that the Corvair wasn't uniquely unsafe

Unsafe at Any Speed (Ralph Nader, 1965)

- Explained how car designers failed to consider safety, and just how bad the results were.
- Picked on the Corvair, in particular.
- Instigated many current safety features.
- Launched Nader's public career.



What can SelfDN-ers learn from auto design?

- Any control system (human or automated) has its limits
- Pushing a control system past its limits can cause crashes
- Sometimes, the problem is not in the control system!
- I'll illustrate with some analogies drawn from *Unsafe at Any Speed*
  - all quotations in these slides are from Ralph Nader, *Unsafe at Any Speed*, New York: Grossman Publishers, 1965, unless otherwise noted

OK, why not build a self-driving Corvair?

Nader's assertion: the Corvair was inherently unsafe

- Even expert drivers found it challenging ("fun") to drive in some conditions
- Average drivers often found themselves in trouble unexpectedly
- Past a certain point, it was impossible to recover
- People were killed or injured unnecessarily
- Various intentional design choices led to these safety problems
- (and Nader alleges bad intentions on the part of GM, but that's not relevant to my talk)

OK, why not build a self-driving Corvair?

Nader's assertion: the Corvair was inherently unsafe

- Even expert drivers found it challenging ("fun") to drive in some conditions
- Average drivers often found themselves in trouble unexpectedly
- Past a certain point, it was impossible to recover
- People were killed or injured unnecessarily
- Various intentional design choices led to these safety problems
- (and Nader alleges bad intentions on the part of GM, but that's not relevant to my talk)

Given these properties, a Corvair would be a poor base for a self-driving car:

- The control system would have to be especially wonderful, or disaster ensues
- The self-driver would be blamed for accidents outside of its control

OK, why not build a self-driving Corvair?

Nader's assertion: the Corvair was inherently unsafe

- Even expert drivers found it challenging ("fun") to drive in some conditions
- Average drivers often found themselves in trouble unexpectedly
- Past a certain point, it was impossible to recover
- People were killed or injured unnecessarily
- Various intentional design choices led to these safety problems
- (and Nader alleges bad intentions on the part of GM, but that's not relevant to my talk)

Given these properties, a Corvair would be a poor base for a self-driving car:

- The control system would have to be especially wonderful, or disaster ensues
- The self-driver would be blamed for accidents outside of its control
- Our networks today are more like 1965 Corvairs than 2018 Volvos.

What was wrong with the Corvair?

Nader alleged the car was far too unstable in cornering manoeuvers:

it "abruptly decides to do the driving for the driver in a wholly untoward manner"

because:

- "swing axle" suspension caused rear wheels to "tuck under" and lose contact
- the unusual rear-heavy weight distribution contributed to this problem
- GM's solution was an unusual/finicky tire-pressure distribution (F=15/R=26 psi)
  - instead of building a slightly more expensive fully independent rear suspension + anti-roll bar

Tire pressure? Really?

Nader writes:

- "Instead of all stability being inherent in the vehicle design, the operator is relied upon to maintain a require [front vs. rear] pressure differential …"
- "any policy which [burdens the driver with monitoring tire pressure differentials] closely and persistently … cannot be described as sound or safe engineering."
- "This responsibility … is passed along to service station attendants, who are notoriously unreliable in abiding by requested tire pressures."
- "There is also serious doubt whether the owner or service man (sic) is fully aware of the importance of maintaining the recommended pressures."
- + "little details" such as the location of the spare tire, and the number of passengers & their luggage, contributed additional complexity

What does this have to do with networks?

- Just like cars, networks can be unstable
- We can't really expect automated control planes to cope with arbitrary instability
  - Just as we shouldn't expect non-expert drivers to cope with unstable cars
- It might be better to fix the instabilities rather than trying to create super AIs
  - Or at least, to clearly define and bound the unstable regimes
- Sometimes, this means moving the stability control "into the network"
  - e.g., today's cars come with traction control, anti-lock brakes, and electronic stability control

Some examples on the next few slides

Some causes of network instability

Multiple control loops trying to optimize the same network:

- E.g., traffic engineering and congestion control working at cross purposes

Too much latency in the control loop:

- E.g., link flaps faster than the routing system can re-converge
- (Maybe the routing plane can handle one flapping link, but not seven at once)

Accidental large-scale synchronization:

- E.g., Sally Floyd and Van Jacobson. 1994. *The synchronization of periodic routing messages*. IEEE/ACM Trans. Netw. 2, 2 (April 1994)

Some causes of network instability

Multiple control loops trying to optimize the same network:

- E.g., traffic engineering and congestion control working at cross purposes

Too much latency in the control loop:

- E.g., link flaps faster than the routing system can re-converge
- (Maybe the routing plane can handle one flapping link, but not seven at once)

Accidental large-scale synchronization (which led to massive packet loss):

- E.g., Sally Floyd and Van Jacobson. 1994. *The synchronization of periodic routing messages*. IEEE/ACM Trans. Netw. 2, 2 (April 1994)

Maybe we should fix such problems directly, rather than asking SelfDN to cope?

Byzantine faults

Corvair example: arguably a "byzantine fault":

- if the rear wheel collapses, the car doesn't steer in the direction the driver chooses

Network controllers have to anticipate byzantine faults, too:

- Routing-system faults that randomly drop, or forward to the wrong destination
- Switches that report bogus link utilizations
  - If I have time at the end of the talk, I'll run through a concrete example
- Routing services that acknowledge new configs but fail to install them

Automation failure vs. hard-to-control networks?

We want to automate network management because operators cause most failures.

But is the problem really the fault of the human operators?

- Well, yes, often it is
- But automation failures are common, too.

Is the problem faulty automation?

- Sometimes yes, but: a hard-to-automate network makes this more likely
- Nader: "**the entire investigatory [apparatus] looks almost invariably to driver failure as the cause [of an accident]**" and argues that this improperly shifts the blame away from the car designer.

Ambiguous/finicky interfaces

Nader: "**The connection between design defects and driver misjudgement … is so subtle that neither the accident investigator nor the driver is aware of this.**"

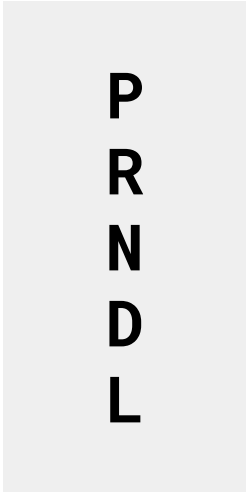
- He attributed many "driver errors" to bad automatic transmission shifter design

We all know the shift pattern for an automatic transmission:

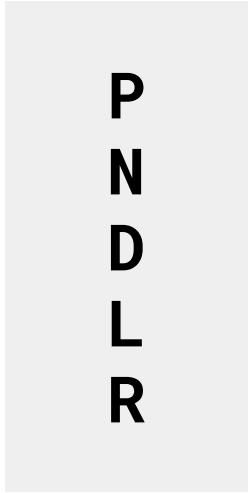
P
R
N
D
L

But some 1960s cars used a different pattern

Familiar pattern:



Atypical pattern:
(slightly cheaper HW)



But some 1960s cars used a different pattern

Familiar pattern:

P
R
N
D
L

Problem #1:

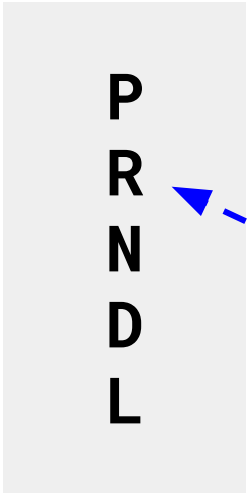
It's too easy to aim for "R" and hit "L" instead:

Atypical pattern:
(slightly cheaper HW)

P
N
D
L
R

But some 1960s cars used a different pattern

Familiar pattern:

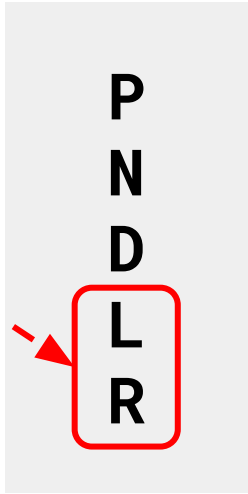


Problem #1:

It's too easy to aim for "R" and hit "L" instead:

The traditional pattern places "R" near between two **non**-forward settings, making this accident much less likely

Atypical pattern:
(slightly cheaper HW)



Example of a metastable API leading to a network outage

Consider this procedure for decommissioning some switches:

1. `Do a DB lookup to find the list of switches meeting a criterion`
2. `Pass this list to an automated decommissioning system`

Suppose that:

- a fault is reported during step 2, causing the procedure to be retried
- however, all of the switches were actually decomm'ed in the first iteration
- so the DB lookup returns an empty list

Example of a metastable API leading to a network outage

Consider this procedure for decommissioning some switches:

1. `Do a DB lookup to find the list of switches meeting a criterion`
2. `Pass this list to an automated decommissioning system`

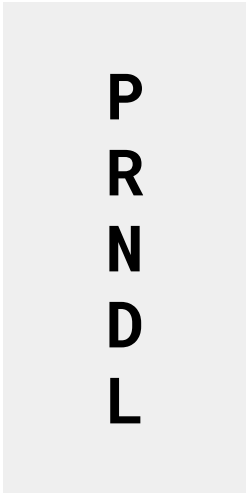
Suppose that:

- a fault is reported during step 2, causing the procedure to be retried
- however, all of the switches were actually decomm'ed in the first iteration
- so the DB lookup returns an empty list
- and the auto-decomm API interprets an empty list to mean "everything"

Mega-oops.

And that wasn't the only problem

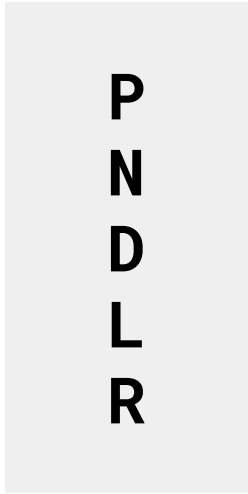
Familiar pattern:



Problem #2:

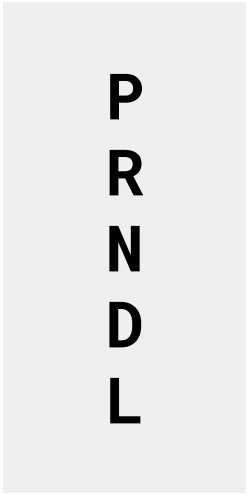
Drivers in two-car families would often get confused and shift the wrong way

Atypical pattern:
(slightly cheaper HW)



And that wasn't the only problem

Familiar pattern:

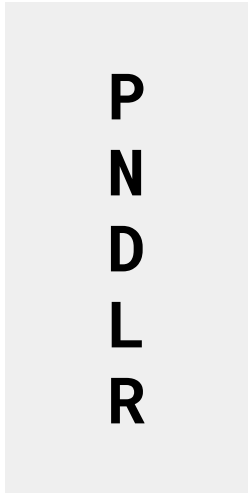


Problem #2:

Drivers in two-car families would often get confused and shift the wrong way

Standardized interfaces avoid "bugs" due to failure to check the context

Atypical pattern:
(slightly cheaper HW)



Missing or imprecise standards

"OpenFlow Needs You! A Call for a Discussion About a Cleaner OpenFlow API"
Peter Peresini, Maciej Kuzniar, and Dejan Kostic, Proc. 2nd European Workshop on SDN, 2013.

points out the dangers of ambiguity at a network-control interface, e.g.:

- a switch's hardware does not support the OpenFlow priority field ...
- … and the specification does not dictate specific behavior in this case
  - (e.g., should the switch allow only single value of priority field?).
- Thus, instead of failing all FlowMod requests with conflicting priorities, the switch silently ignores this field.

Too many "standards"

Cisco uses imperative firewall rules:

```
permit tcp host 172.21.1.1 host 172.21.1.15 eq 443
```

Juniper uses "object-oriented" rules:

```
filter 1 { term T1 { from { source-address { 172.21.1.1/32; } destination-address {
172.21.1.15/32; } protocol tcp; destination-port 443;} then {accept; }}}
```

It's non-trivial to write code that translates the same intent to both syntaxes:

- Liu, Holden, Wu. Automated Synthesis of Access Control Lists. In *Proc. 3rd IEEE Intl. Conf. on Software Security and Assurance* (2017).
- It's especially **hard to translate between standards**, because extracting "intent" is tricky

Too many incompatible standards?

- Create another one -- OpenConfig FTW!

Too many "standards"

Cisco uses ii

```
permit t
```

Juniper uses

```
filter 1
172.21.1
```

```
on-address {
```

It's non-trivia

- Liu, Holde
- It's espec

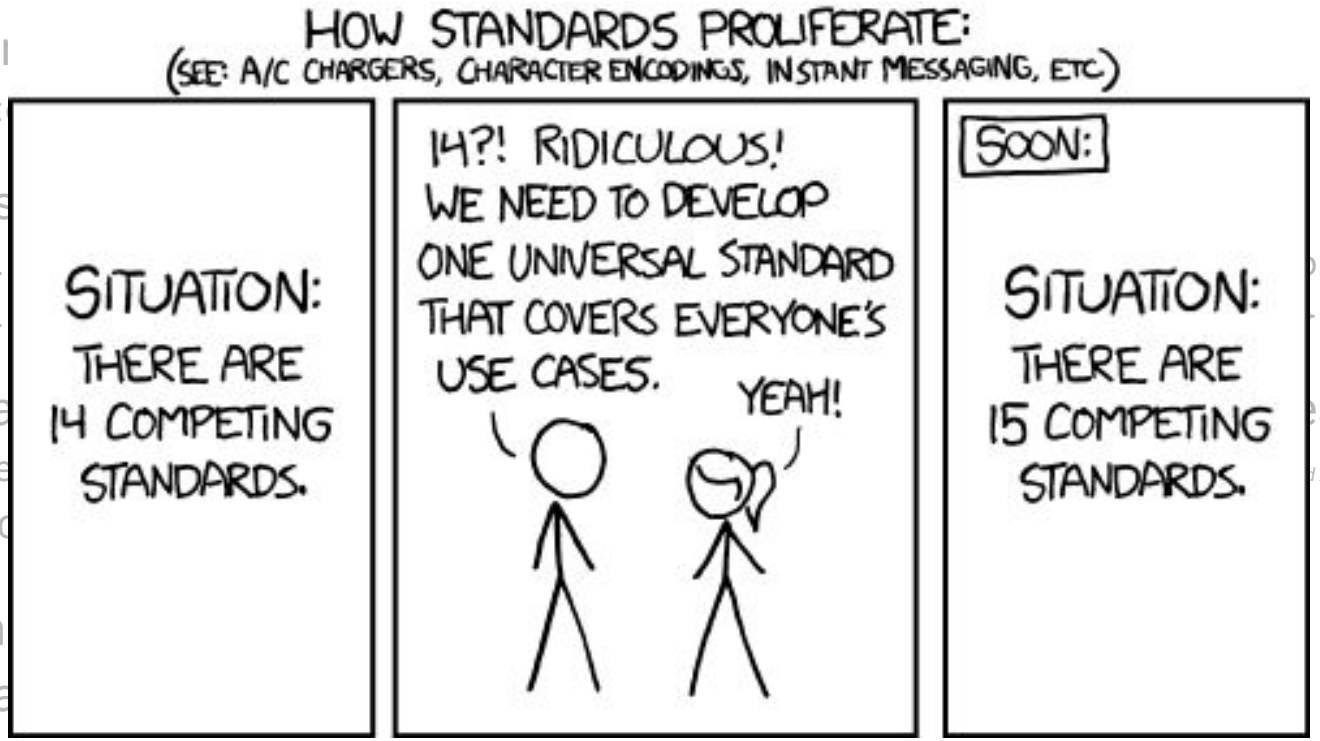
l Assurance (2017).

es:

is tricky

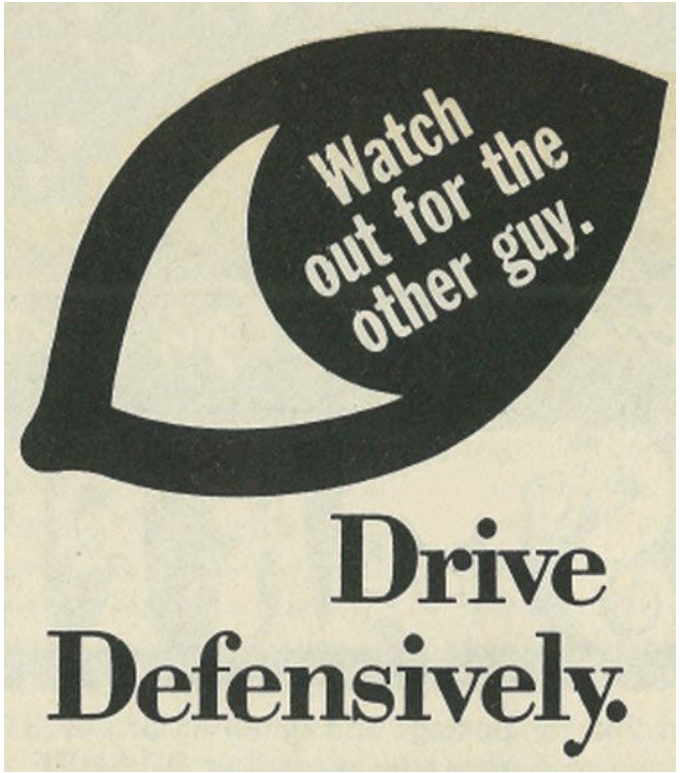
Too many in

- Create a



xkcd.com/927/

Not all crashes are solo crashes



1971 Public Service Ad

Older cars made it too hard to "watch out for the other guy"

"Until side view mirrors were mandated for safety reasons in the 1960s, they were often not a standard item. Many buyers resisted shelling out an extra $15 or so for an extra mirror." https://www.ebay.com/motors/blog/automotive-side-view-mirrors/

- Right-side mirrors were optional until relatively recently

Older cars made it too hard to "watch out for the other guy"

"Until side view mirrors were mandated for safety reasons in the 1960s, they were often not a standard item. Many buyers resisted shelling out an extra $15 or so for an extra mirror." https://www.ebay.com/motors/blog/automotive-side-view-mirrors/

- Right-side mirrors were optional until relatively recently

Fun facts:

1. *The first rear-view mirror was invented by racer Ray Harroun, who figured he could save 160 lbs by not having a passenger to tell him if it was safe to pass.*
    - *But the 1911 Indy 500 cobbled race track shook the mirror too much, and it was useless*
2. *The 1921 patented product was marketed as a "cop spotter"*
    - *Possibly true: which led some states to ban rear mirrors*

SelfDNs also need to "Watch out for the other ~~guy~~ operator"

Network control operations have to be coordinated with:

- Power-system maintenance
- Rack moves, broken rack wheels, and other facility maintenance
- Peak workloads from Cloud customers
- "Federated" network managers (sometimes less than cooperative):
  - WAN fiber repairs (including incorrect cabling)
  - Configuration of peering links
  - Accidental BGP hijacks
  - etc.

"No network is an island" -- your SelfDN control plane isn't, either

Prevent accidents, or survive them?

Nader: **even experts "believed that forces involved in [auto or air] crashes were too severe for the human body to absorb, [so] they concentrated on preventing accidents rather than preventing injuries when accidents do occur."**

Thus, they ignored the "second collision" when the passenger hits the car:

- Rigid steering columns impaling the driver
- Rigid dashboards, pointy instrument knobs, and style features that maimed
- Gruesome effects of hitting the windshield glass
- Crushed by a collapsed roof
- Being ejected from relative safety of the car
- etc.

The "second collision" in network failures

Post-mortems of our outages often reveal failures are the result of multiple faults

- For example, a control plane can handle single failures, but not lots at once
- Or a low-level fault triggers incompletely-tested recovery code in control plane
- Or both: recovery code works with single failures, but not lots of them

Fault-recovery code can sometimes cause failure due to resource exhaustion, e.g.:

1. Control plane detects data-plane problem that it could route around
2. Control plane dutifully sends reports to remote logger … but too many of them
3. Un-sent log reports pile up in buffer at controller … unbounded.
4. Controller dies because it has no remaining memory

The "second collision" in network failures

Post-mortems of our outages often reveal failures are the result of multiple faults

- For example, a control plane can handle single failures, but not lots at once
- Or a low-level fault triggers incompletely-tested recovery code in control plane
- Or both: recovery code works with single failures, but not lots of them

Fault-recovery code can sometimes cause failure due to resource exhaustion, e.g.:

1. Control plane detects data-plane problem that it could route around
2. Control plane dutifully sends reports to remote logger … but too many of them
3. Un-sent log reports pile up in buffer at controller … unbounded.
4. Controller dies because it has no remaining memory

Today, we survive car crashes, not just prevent them

- Well-anchored, comfortable 3-point seatbelts
- Airbags
- Well-padded dashboards
- No sharp objects
- Head restraints on the front seats

Self-Driving networks: design them for safe driving

A car has "caster" so that when you release the wheel, it returns to a straight path.

An "aerodynamically stable" aircraft tends to fly (relatively) straight and level if you let go of the controls.

A typical sailboat will turn into the wind and stop if you let go of the tiller.

- Or with multiple sails and "balanced" trim, it will keep going (relatively) straight

Self-Driving networks: design them for safe driving

A car has "caster" so that when you release the wheel, it returns to a straight path.

An "aerodynamically stable" aircraft tends to fly (relatively) straight and level if you let go of the controls.

A typical sailboat will turn into the wind and stop if you let go of the tiller.

- Or with multiple sails and "balanced" trim, it will keep going (relatively) straight

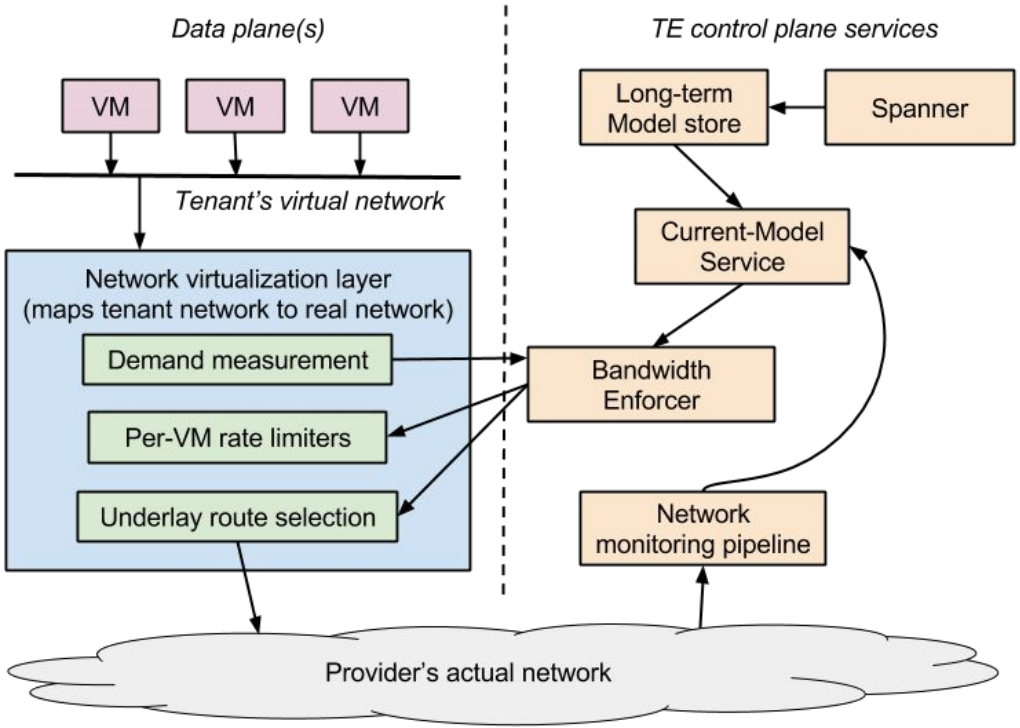
We can do the same for networks:

- e.g., if the centralized control plane fail-stops, local control continues to "fly straight and level" -- e.g., via fallback from centralized TE to ISIS

A concrete example: Traffic Engineering with *Bandwidth Enforcer*

Bandwidth Enforcer (BwE):

- Receives demand measurements from hosts
- Learns current network link states and utilizations
- Computes admissible traffic
- Asks hosts to throttle senders accordingly

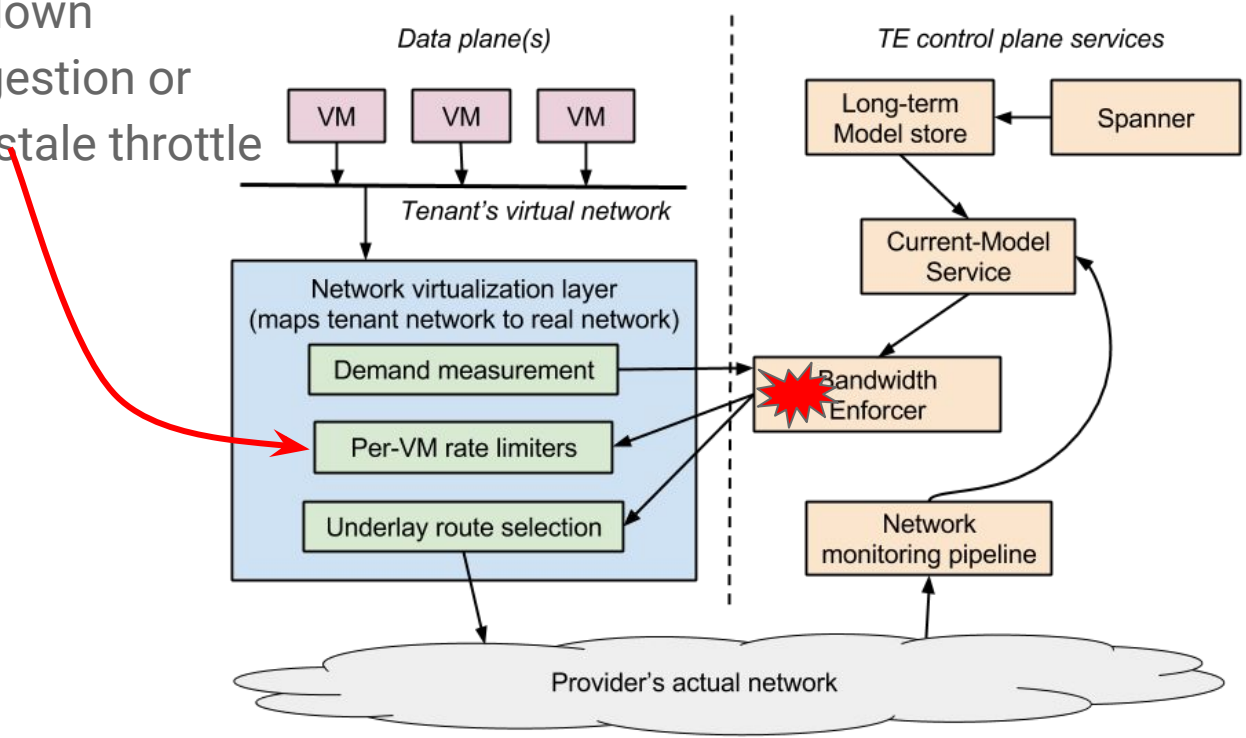


Kumar *et al.*, BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. SIGCOMM '15.

Fail-static with BwE

VMs keep running if BwE is down

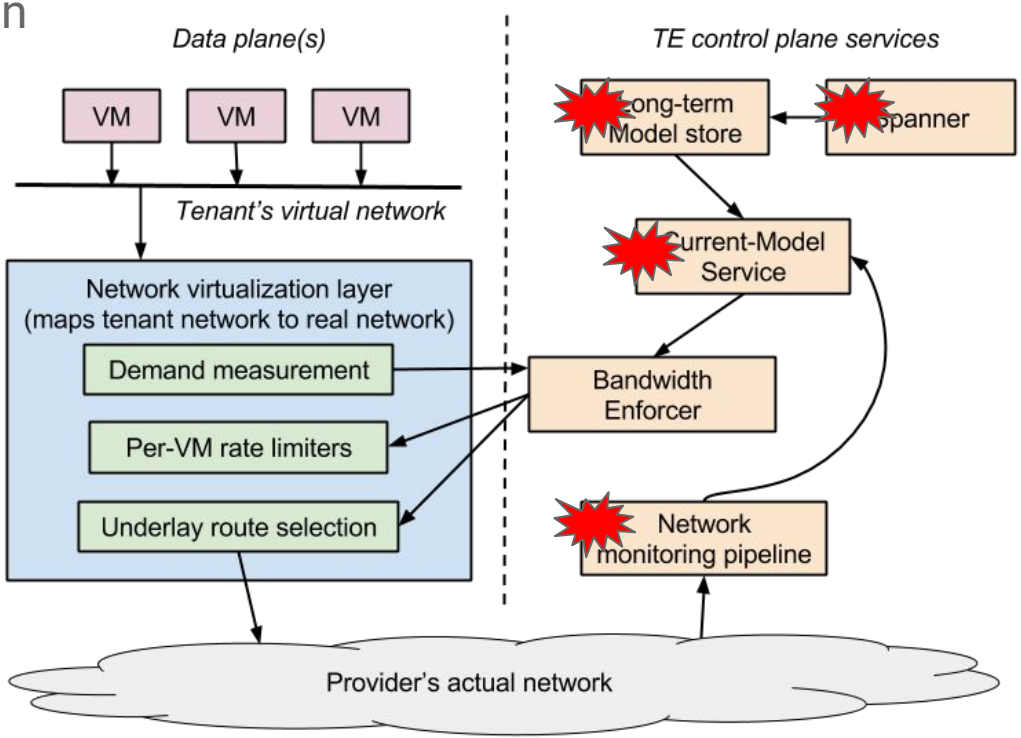
- but with increasing congestion or under-utilization, due to stale throttle settings



Fail-static with BwE

BwE keeps running if monitoring is down (in any of several components)

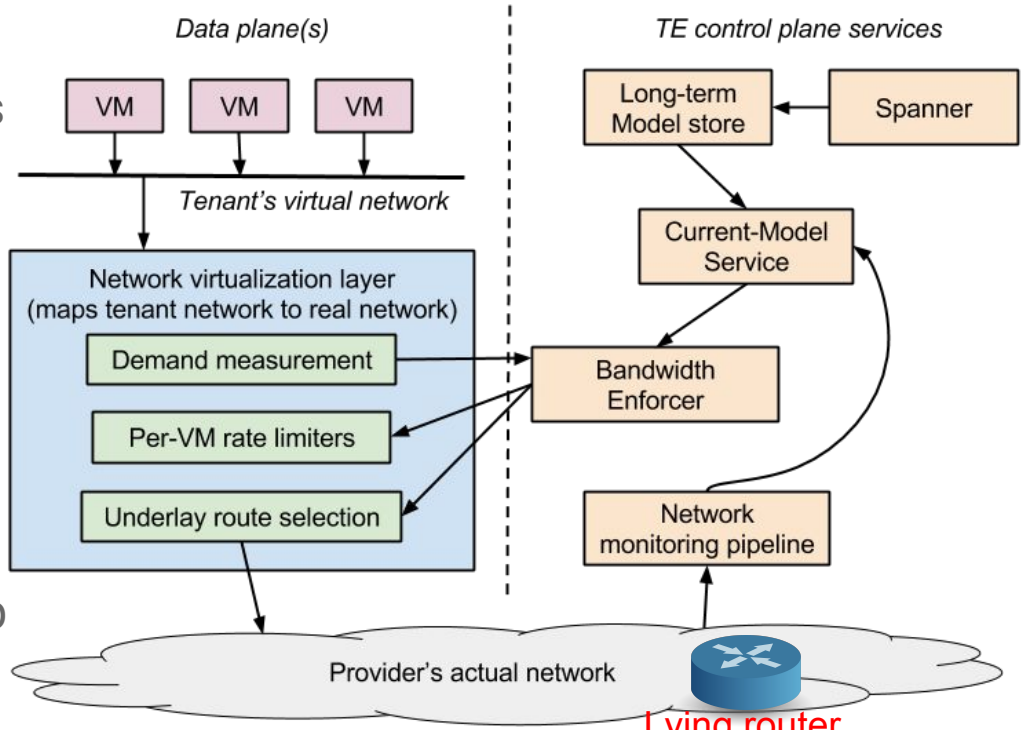
- but might have to use increasingly conservative throttling, due to increasingly stale view of network



Why BFT doesn't necessarily work for BwE

- Lying router falsely reports 100% utilized link
- Monitoring stack faithfully conveys this "alternative fact" to BwE
- BwE throttles the sources that are using this link
- Reported utilization doesn't drop!
- Process repeats until no traffic flows

Where could BFT (*3f+1* replication) help with this Byzantine failure?



The problem of maps

Self-driving cars depend on highly-detailed, accurate, up-to-date maps

- "the most detailed and expansive version of these maps ... could be worth billions of dollars." *(Mark Bergen, Bloomberg, 2018-04-13)*
- Even if 1965 car had supported self-driving, the 1965 maps could not have
- But: good-enough maps exist today, for many areas

Self-driving networks also need good "maps"

- i.e., detailed representations of network topology & state
- Without such data, it's hard to operate manually, but impossible to automate
- It's really hard to get detailed/accurate/up-to-date network "maps" today
- … that's the topic of another long talk!

Don't forget testing

Nader: **"The combination of factors with leads to the critical point of control loss may occur with a statistical infrequency. [Therefore,] stability limitations … must be evaluated under the most unfavorable loading conditions."**

Testing self-driving systems

Without a human operator, pre-deployment testing has to anticipate all kinds of stuff

- Vint Cerf, *A Comprehensive Self-Driving Car Test*, CACM Feb 2018: "***We've staged people jumping out of canvas bags or portable toilets on the side of the road, skateboarders lying on their boards [etc.]***"

If you're building a SelfDN, you have the responsibility to worry about testing it

- What's your equivalent of the person jumping out in front of the car?
- How do you **know** you've thought about all of the weird things & black swans?

Personal opinion: a SelfDN paper without a discussion of testing is … unconvincing.

The value of blameless postmortems

Blameless postmortems:

- Analyze and fix the systems and the environment that led to a failure
- Don't blame the people who happened to be present when it all went wrong
- Blame and honest analysis seldom go well together

SelfDN success will depend on "**fixing the environment**," just as much as on great ML

Note that Nader's book vigorously assigned blame, but he had a different agenda

- In fact, he scorned "fix-the-roads" proposals that distracted from improving cars

I hope I've scared you a bit

… but also I hope I've encouraged you to

- look at how the requirements for designing a self-driving control plane interact with the design of the underlying network.
- think about focusing some effort into designing networks that are
  - easy to understand
  - easy to control
  - easy to verify
  - stable when the control plane is down

As Jen said: Fight to enable SelfDN … don't just through ML at infeasible problems