#eBPF A Wholistic Approach

David S. Miller, Red Hat Inc.

davem_dokebi

Or……

Inner Peace With #eBPF

David S. Miller
Red Hat Inc.

davem_dokebi

What Is Awesome Now

The Concept

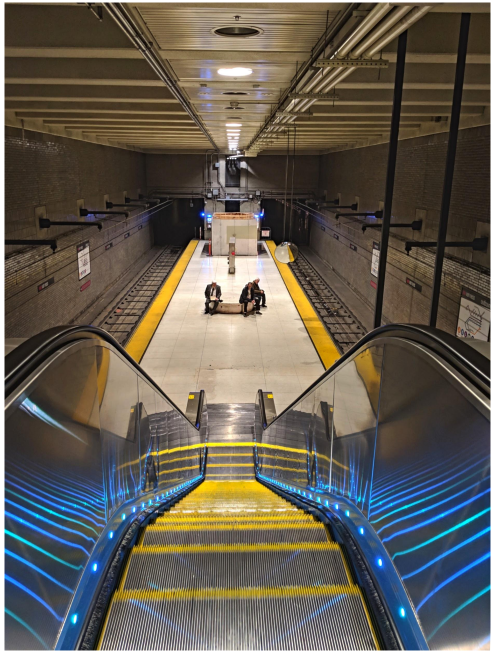
At a high level #ebpf is well defined and useful

It solves a real problem space

Making the kernel do what we want

Inspecting the kernel's actions

All without core modifications or rebooting



The Environment

Well defined execution environment with well defined limits

Programs operate on “objects”

Programs maintain state using MAPS

No endless loops, execute in finite amount of time

Verifier can look for “crap” or “sneaky stuff”

The Change In The Power Dynamic

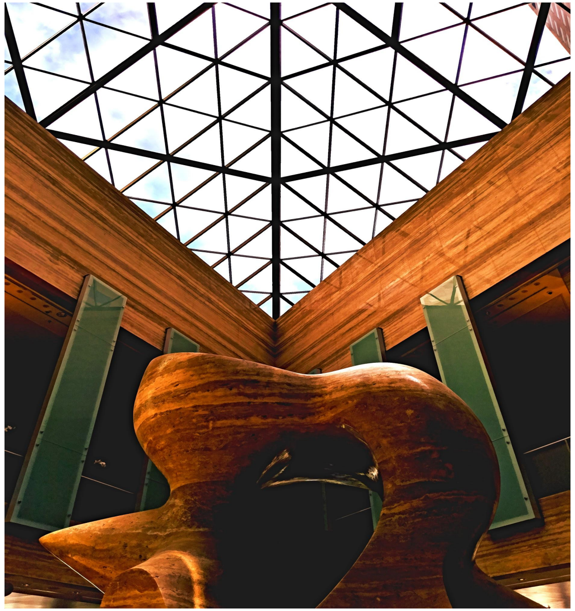
System designers are arrogant and build “boxes”

Users don’t want to be in a box

#ebpf breaks them free from the usual confines

Users now can control their systems, their way

System designers are sad, but they will get over it



We Made An Impression

People are really excited about the technology

It is so good, it becomes self-advertising

Techies who catch the bug tell their friends

Word is spreading fast in the container world thanks to Cilium

People are... writing books

What Is Improving

Ability To Write “Real Programs”

Small simple programs are a conquered space

Large programs are the next big area

Increase in allowed program size is a big improvement here

As is the ability to use bounded loops

Combining tail calls and real calls

Introspection

I actually start to hate this word, it means too many things

BTF will allow debuggability and “write mostly-once” code

bpftool is improving and becoming more powerful all the time

Perf can fully annotate #ebpf programs (looks amazing)

What We Need

Reusable Code

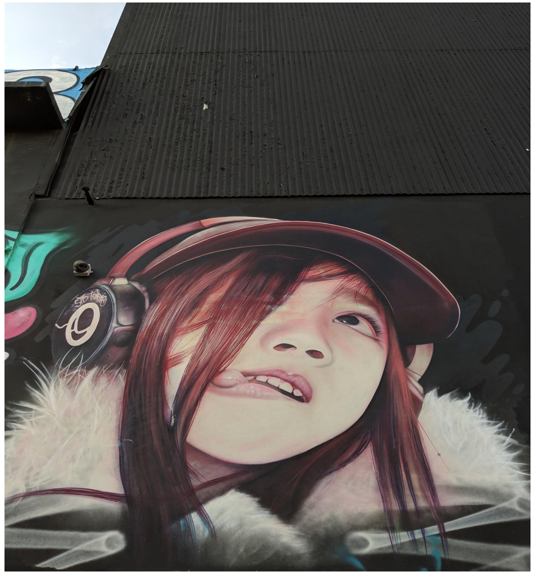
1,000 copies of ethernet header parsing is **not** the future

We have real function calls but…

… we need real libraries in #ebpf

Design issue: Deciding library ownership and access control

Maybe “global” access for some and cooked “libs” in kernel image



Type And Go Development

Easier to write #ebpf programs now than before for sure

However we can of course improve, think Arduino

I want anyone to be able to code simple #ebpf programs

#ebpf is “programming for the people”

Let’s help them get there

Trivial Debuggability

I have to be able to “see” exactly what the program does

And also all of the data it operates upon

Triggerable snapshots of single-step execution

Along with a copy of the context data

Sent perhaps as a perf event or similar

Better Access Control

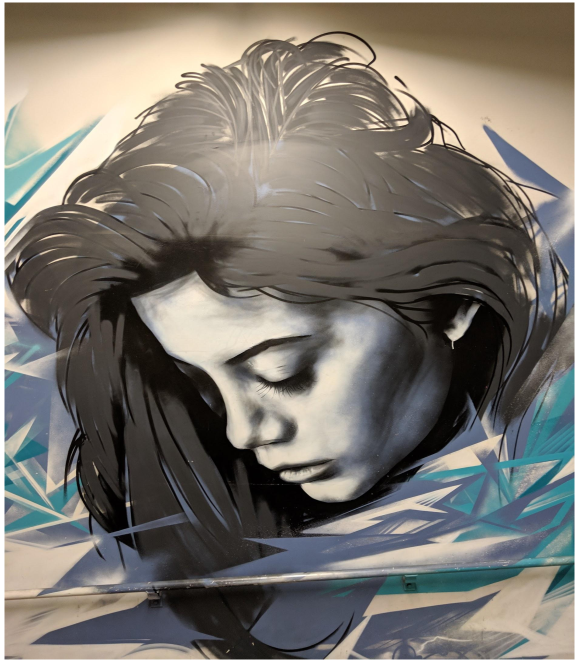
Root vs. non-root, not enough granularity

“/dev/bpf” file, if you can open you can use #ebpf

Or a hierarchy of files, #ebpf sub-permissions

Issues of shared FD tables and permission inheritance

Memory accounting (prog + MAPS)



Questions?