eBPF 101

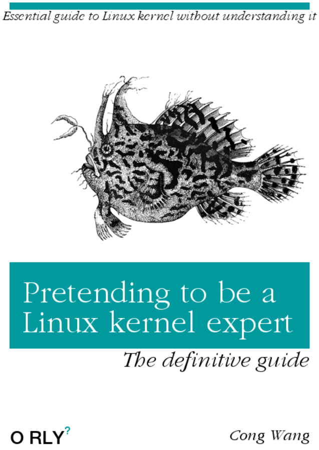
An overview from a perspective of a non-kernel programmer

Muhammad Falak R Wani
falakreyaz@gmail.com

Linux Conf Au 2022

whoami[1]

I admit I am an imposter[2]



Essential guide to Linux kernel without understanding it

Pretending to be a Linux kernel expert

The definitive guide

O RLY?

Cong Wang

[1] mfrw.github.io

[2] Pretending to be a Linux Kernel Expert

Agenda

1. History
   - Motivation
   - Problem
   - Solution – BPF
2. Foundations
   - eBPF Architecture
   - eBPF Prog Types
   - eBPF Maps
3. Conclusion

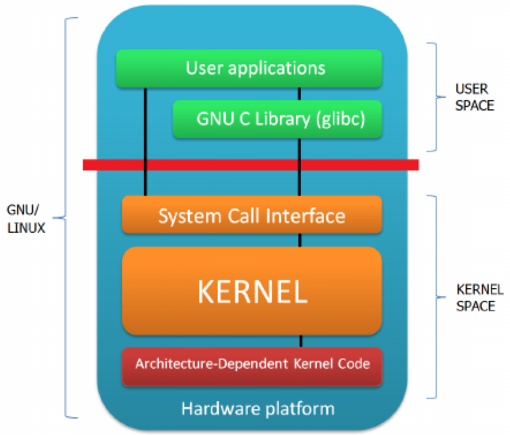
Agenda

1. History
   - Motivation
   - Problem
   - Solution – BPF
2. Foundations
   - eBPF Architecture
   - eBPF Prog Types
   - eBPF Maps
3. Conclusion

Let's design a Packet Filter

How hard can that be?

- Copy everything to user-space
- Write a Kernel Module



What is the Problem?

User Space vs Kernel Space Trade Off

User Space

- We copy every packet to User Space which is not the most optimal solution.
- Copies everything.
- Not Optimal Performance.
- *Generic Solution.*
- SAFE – SEGFAULT.

Kernel Space

- Hardcoding what packets that we are interested in is not a generic solution
- Copy only what we want.
- *Optimal Performance.*
- Not a Generic Solution.
- UNSAFE – System down.

What is the Problem?

User Space vs Kernel Space Trade Off

User Space

- We copy every packet to User Space which is not the most optimal solution.
- Copies everything.
- Not Optimal Performance.
- *Generic Solution.*
- SAFE – SEGFAULT.

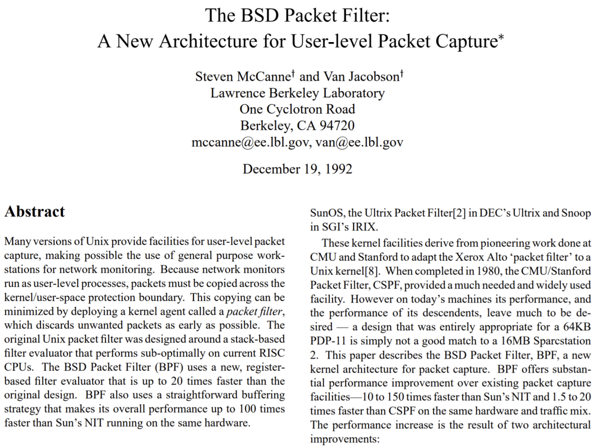
Kernel Space

- Hardcoding what packets that we are interested in is not a generic solution
- Copy only what we want.
- *Optimal Performance.*
- Not a Generic Solution.
- UNSAFE – System down.

> What if we had best of both the worlds ?
>
> — Anonymous Engineer

BPF

A seminal paper published in 1992



The BSD Packet Filter: A New Architecture for User-level Packet Capture[*]

Steven McCanne[†] and Van Jacobson[†]
Lawrence Berkeley Laboratory
One Cyclotron Road
Berkeley, CA 94720
mccanne@ee.lbl.gov, van@ee.lbl.gov

December 19, 1992

Abstract

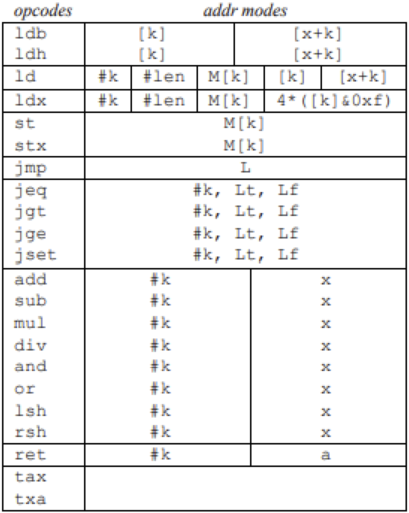
Many versions of Unix provide facilities for user-level packet capture, making possible the use of general purpose workstations for network monitoring. Because network monitors run as user-level processes, packets must be copied across the kernel/user-space protection boundary. This copying can be minimized by deploying a kernel agent called a *packet filter*, which discards unwanted packets as early as possible. The original Unix packet filter was designed around a stack-based filter evaluator that performs sub-optimally on current RISC CPUs. The BSD Packet Filter (BPF) uses a new, register-based filter evaluator that is up to 20 times faster than the original design. BPF also uses a straightforward buffering strategy that makes its overall performance up to 100 times faster than Sun's NIT running on the same hardware.

SunOS, the Ultrix Packet Filter[2] in DEC's Ultrix and Snoop in SGI's IRIX.

These kernel facilities derive from pioneering work done at CMU and Stanford to adapt the Xerox Alto 'packet filter' to a Unix kernel[8]. When completed in 1980, the CMU/Stanford Packet Filter, CSPF, provided a much needed and widely used facility. However on today's machines its performance, and the performance of its descendents, leave much to be desired — a design that was entirely appropriate for a 64KB PDP-11 is simply not a good match to a 16MB Sparcstation 2. This paper describes the BSD Packet Filter, BPF, a new kernel architecture for packet capture. BPF offers substantial performance improvement over existing packet capture facilities—10 to 150 times faster than Sun's NIT and 1.5 to 20 times faster than CSPF on the same hardware and traffic mix. The performance increase is the result of two architectural improvements:

BPF

A simple virtual machine residing in the kernel



opcodes	*addr modes*				
ldb	[k]	[x+k]			
ldh	[k]	[x+k]			
ld	#k	#len	M[k]	[k]	[x+k]
ldx	#k	#len	M[k]	4*([k]&0xf)	
st	M[k]				
stx	M[k]				
jmp	L				
jeq	#k, Lt, Lf				
jgt	#k, Lt, Lf				
jge	#k, Lt, Lf				
jset	#k, Lt, Lf				
add	#k	x			
sub	#k	x			
mul	#k	x			
div	#k	x			
and	#k	x			
or	#k	x			
lsh	#k	x			
rsh	#k	x			
ret	#k	a			
tax					
txa					

How Does a BPF program work?

Let's take a digression first

How do userspace programs work ?

Compiled

Write Code → Compiler + Linker → Run the Binary

Interpreted

Write Code → Interpreter → JIT instruction → execute JIT-ed instructions

---

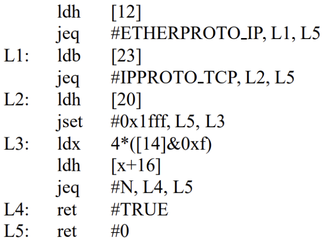
But wait the BPF VM is in the kernel!

BPF

How does a BPF program run ?

BPF a.k.a Classical BPF (cBPF) programs are STATELESS.
Hook points are only in the Network Stack.

- Write a simple program (Filter) using the ISA.
- Filter expressions return True/False.
- Load the ByteCode program in the kernel.
- Attach the loaded program to a hook. (e.g on every received packet)
- Programs are event driven and are run to completion when the event occurs.



```
        ldh     [12]
        jeq     #ETHERPROTO_IP, L1, L5
L1:     ldb     [23]
        jeq     #IPPROTO_TCP, L2, L5
L2:     ldh     [20]
        jset    #0x1fff, L5, L3
L3:     ldx     4*([14]&0xf)
        ldh     [x+16]
        jeq     #N, L4, L5
L4:     ret     #TRUE
L5:     ret     #0
```

Load Byte Code → Interpreter → Attach to Hook → Run BPF → Action

Ideas Similar to BPF

Do not map 1:1 exactly – similar theme[3]

- Embeded lua VM in nginx to modify behaviour without recompiling nginx or writing C.
- Embeded lua VM in neovim to write plugins and extend functionality.
- Vimscript for VIM to extend functionality.
- Writing WebAssembly filters for envoy proxy.
- WebAssembly for browsers.

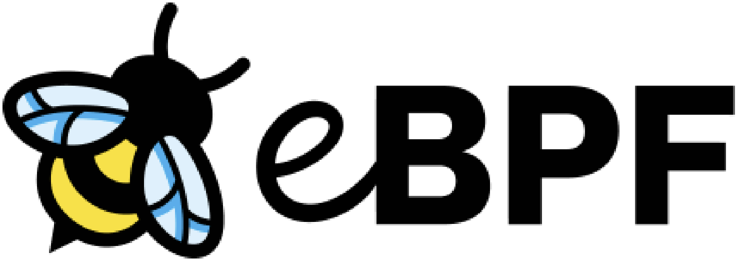
[3] None of them have a verifier

Agenda

1. History
   - Motivation
   - Problem
   - Solution – BPF
2. Foundations
   - eBPF Architecture
   - eBPF Prog Types
   - eBPF Maps
3. Conclusion

eBPF Mascot

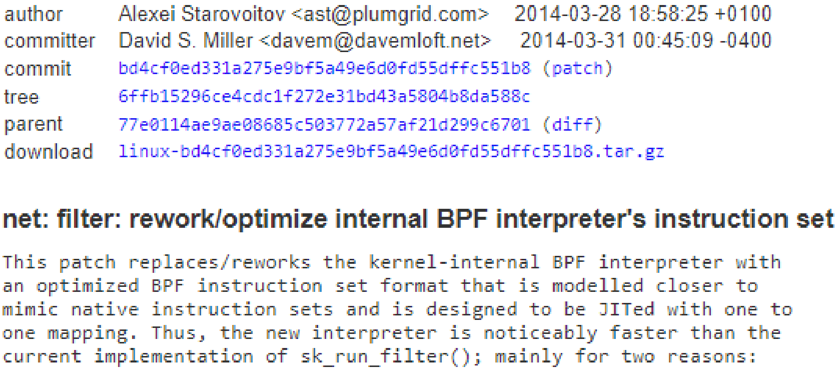
The cute Bee!



extended BPF

BPF VM in the Linux Kernel got improved vastly

Alexei Starovoitov sent a patch improving the existing BPF infrastructure in the kernel and as a result BPF → eBPF.



author	Alexei Starovoitov <ast@plumgrid.com>	2014-03-28 18:58:25 +0100
committer	David S. Miller <davem@davemloft.net>	2014-03-31 00:45:09 -0400
commit	bd4cf0ed331a275e9bf5a49e6d0fd55dffc551b8 (patch)	
tree	6ffb15296ce4cdc1f272e31bd43a5804b8da588c	
parent	77e0114ae9ae08685c503772a57af21d299c6701 (diff)	
download	linux-bd4cf0ed331a275e9bf5a49e6d0fd55dffc551b8.tar.gz	

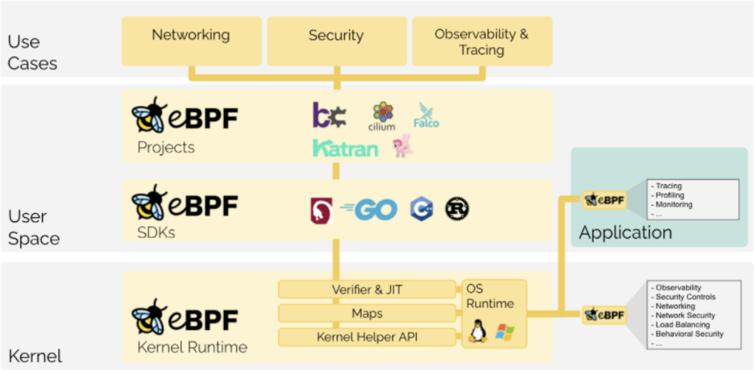
net: filter: rework/optimize internal BPF interpreter's instruction set

```
This patch replaces/reworks the kernel-internal BPF interpreter with
an optimized BPF instruction set format that is modelled closer to
mimic native instruction sets and is designed to be JITed with one to
one mapping. Thus, the new interpreter is noticeably faster than the
current implementation of sk_run_filter(); mainly for two reasons:
```

extended BPF

eBPF is not limited to the network stack[4]

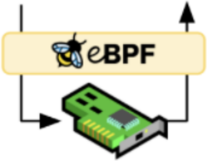
Recall cBPF had hooks only in the network stack. eBPF has hook points all throughout the kernel.



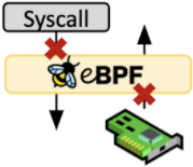
[4] Image Courtesy: ebpf.io

eBPF Capabilities

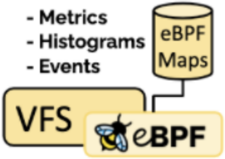
Having a secure VM in the kernel has endless possibilities [5]



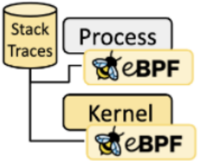
Networking



Security



Observability



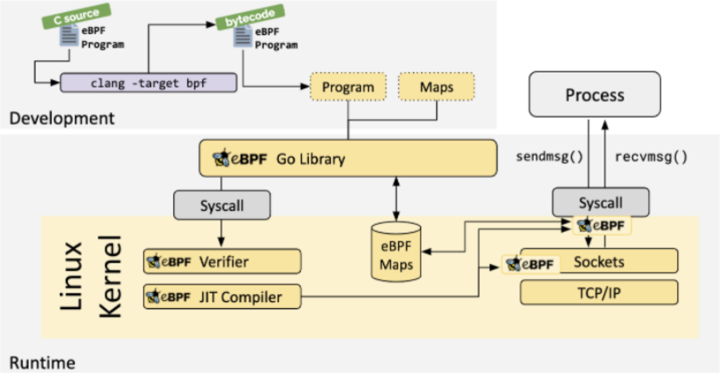
Tracing

[5] Image Courtesy: ebpf.io

eBPF Verifier & JIT

Loading and Attaching a eBPF program[6]

The bpf() syscall is a multi-tool which lets us load & attach an eBPF program.



[6] Image Courtesy: ebpf.io

eBPF Program Types

Different kinds of eBPF programs

A non exhaustive list of BPF_PROG_*:

- BPF_PROG_TYPE_SOCKET_FILTER: a packet filter
- BPF_PROG_TYPE_XDP: a packet filter run from device driver rx path
- BPF_PROG_TYPE_KPROBE: if a kprobe should fire or not
- BPF_PROG_TYPE_TRACEPOINT: if a tracepoint should fire or not
- BPF_PROG_TYPE_SOCK_OPS: set socket options
- BPF_PROG_....

eBPF MAPs

Saving State in eBPF Programs

Recall cBPF was entirely stateless. eBPF is stateless but has the capability to access storage which are called eBPF MAPs. eBPF MAP is a generic data structure that allows data to be passed back and forth withing the kernel or between the user space and the kernel.
eBPF MAPS are created by the same bpf() syscall.

A few interesting BPF_MAP_TYPE_*:

- BPF_MAP_TYPE_HASH: an actual hash table
- BPF_MAP_TYPE_ARRAY: an array
- BPF_MAP_TYPE_PROG_ARRAY: an array of fd's corresponding to eBPF programs.
- BPF_MAP_TYPE_...

Agenda

1. History
   - Motivation
   - Problem
   - Solution – BPF
2. Foundations
   - eBPF Architecture
   - eBPF Prog Types
   - eBPF Maps
3. Conclusion

Conclusion

- eBPF programs run in response to events.
- eBPF programs run to completion.
- Running an eBPF program is much safer than running and maintaining a kernel-module.
- The entry bar to get useful information from the kernel is significantly reduced.
- The overhead of observability is applicable only when you run dynamic instrumentation.

Thank You!