Past, Present, and Future

Feng Li (李枫)
hkli2013@126.com
Sep 22, 2022

Agenda

I. The birth of eBPF
- Kernel development
- cBPF
- eBPF

II. eBPF in today's Linux Kernel
- Kernel subsystems with eBPF

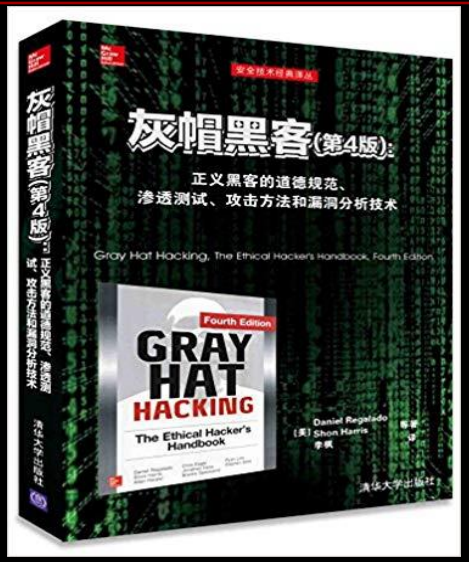
III. eBPF ecosystem
- What's new
- Development
- Applications

IV. The future of eBPF
- Programming
- Deeply integration into the Cloud-native
- eBPF for Edge Computing

V. Wrap-up

Who Am I

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)



- Pure software developmement for ~15 years(~11 years on Mobile Dev)
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

I. The birth of eBPF

1) Kernel development

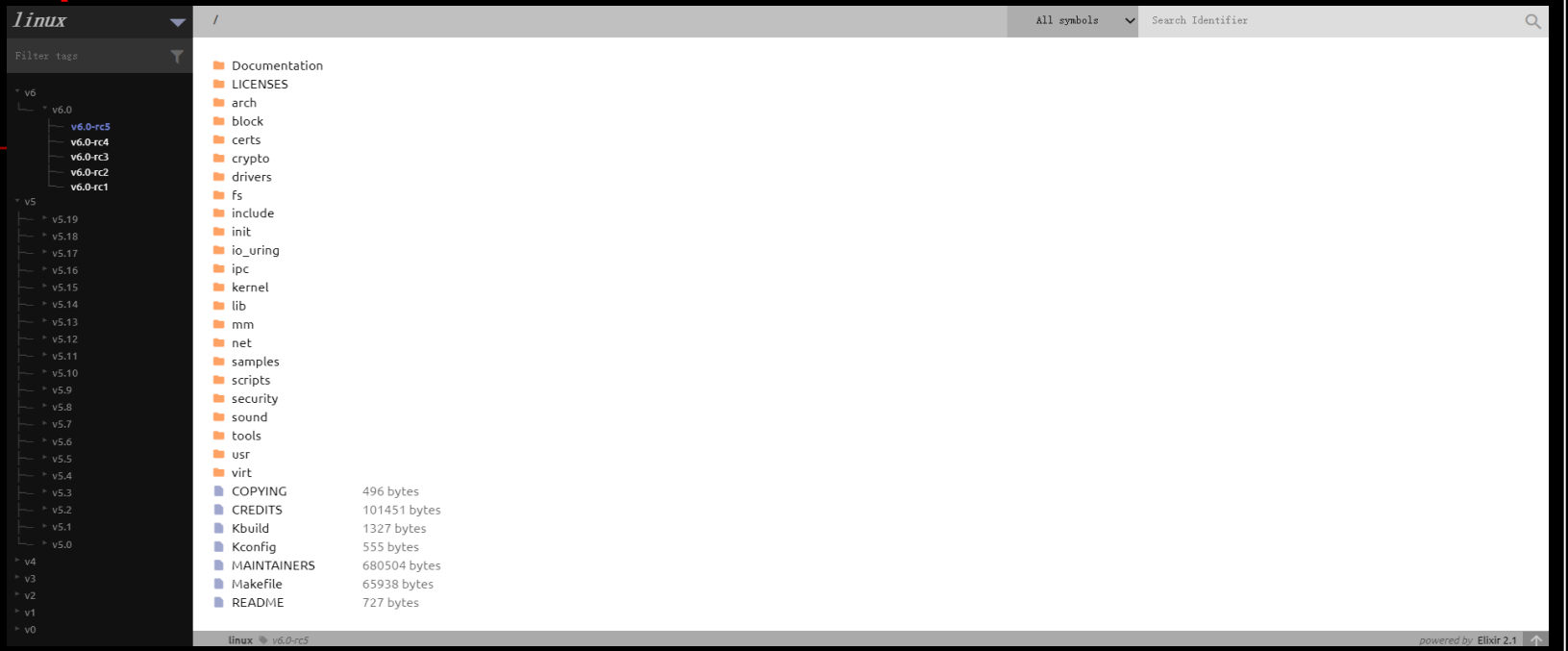
1.1 Overview

Documents

- $KERNEL_SRC/Documentation/kbuild/modules.rst
- $KERNEL_SRC/Documentation/trace/events.rst
- $KERNEL_SRC/Documentation/trace/ftrace*.rst
- $KERNEL_SRC/Documentation/kprobes
- $KERNEL_SRC/Documentation/trace/kprobetrace.rst
- $KERNEL_SRC/Documentation/trace/uprobetracer.rst
- $KERNEL_SRC/Documentation/trace/tracepoints*.rst
- $KERNEL_SRC/Documentation/trace/features/*.*
- …

Browsing Kernel Code

- https://elixir.bootlin.com/linux/latest/source



- https://github.com/torvalds/linux
- https://www.kernel.org/
- https://linuxhint.com/browse_linux_kernel_source/
- …

1.2 DTrace

- https://en.wikipedia.org/wiki/DTrace

DTrace is a comprehensive dynamic tracing framework originally created by Sun Microsystems for troubleshooting kernel and application problems on production systems in real time. Originally developed for Solaris, it has since been released under the free Common Development and Distribution License (CDDL) in OpenSolaris and its descendant illumos, and has been ported to several other Unix-like systems.

DTrace can be used to get a global overview of a running system, such as the amount of memory, CPU time, filesystem and network resources used by the active processes. It can also provide much more fine-grained information, such as a log of the arguments with which a specific function is being called, or a list of the processes accessing a specific file.

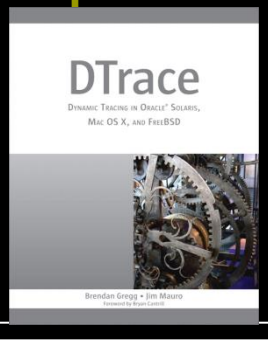
...

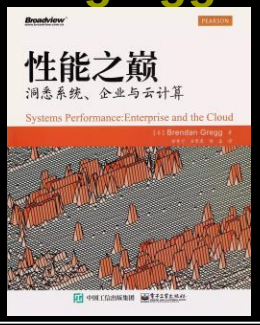
In October 2011, Oracle announced the porting of DTrace to Linux,[4] and in 2019 official DTrace for Fedora is available on GitHub. For several years an unofficial DTrace port to Linux was available, with no changes in licensing terms.[5]

In August 2017, Oracle released DTrace kernel code under the GPLv2+ license, and user space code under GPLv2 and UPL licensing.[6] In September 2018 Microsoft announced that they had ported DTrace from FreeBSD to Windows.[2]

In September 2016 the OpenDTrace effort began on github with both code and comprehensive documentation of the system's internals. The OpenDTrace effort maintains the original CDDL licensing for the code from OpenSolaris with additional code contributions coming under a BSD 2 Clause license. The goal of OpenDTrace is to provide an OS agnostic, portable implementation of DTrace that is acceptable to all consumers, including macOS, FreeBSD, OpenBSD, NetBSD, and Linux as well as embedded systems.

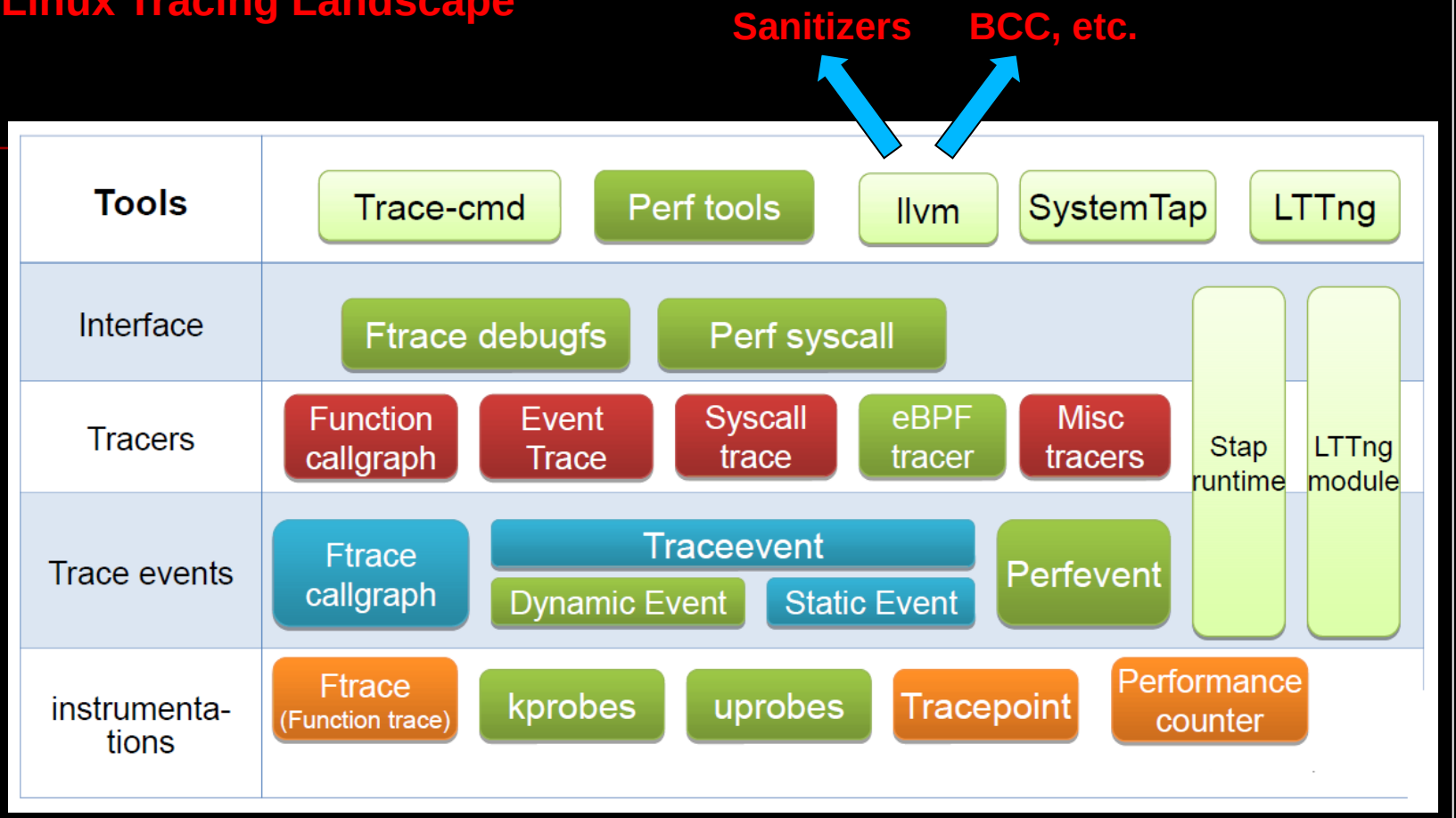
- https://docs.oracle.com/en/operating-systems/oracle-linux/dtrace-guide
- http://www.brendangregg.com/dtrace.html





1.3 Linux World

- Linux Tracing Landscape



Source: "Dynamic Probes for Linux", Masami Hiramatsu, Tracing Summit 2015.

1.3.1 Systemtap

- https://en.wikipedia.org/wiki/SystemTap

In computing, **SystemTap** (stap) is a scripting language and tool for dynamically instrumenting running production Linux-based operating systems. System administrators can use SystemTap to extract, filter and summarize data in order to enable diagnosis of complex performance or functional problems.

SystemTap consists of free and open-source software and includes contributions from Red Hat, IBM, Intel, Hitachi, Oracle, the University of Wisconsin-Madison and other community members.[1]

Usage [edit]

SystemTap files are written in the SystemTap language[7] (saved as `.stp` files) and run with the `stap` command-line.[8]

The system carries out a number of analysis passes on the script before allowing it to run. Scripts may be executed with one of three backends selected by the `--runtime=` option. The default is a loadable kernel module, which has the fullest capability to inspect and manipulate any part of the system, and therefore requires most privilege. Another backend is based on the dynamic program analysis library DynInst to instrument the user's own user-space programs only, and requires least privilege. The newest backend[9] is based on eBPF byte-code, is limited to the Linux kernel interpreter's capabilities, and requires an intermediate level of privilege. In each case, the module is unloaded when the script has finished running.

Scripts generally focus on events (such as starting or finishing a script), compiled-in probe points such as Linux "tracepoints", or the execution of functions or statements in the kernel or user-space.

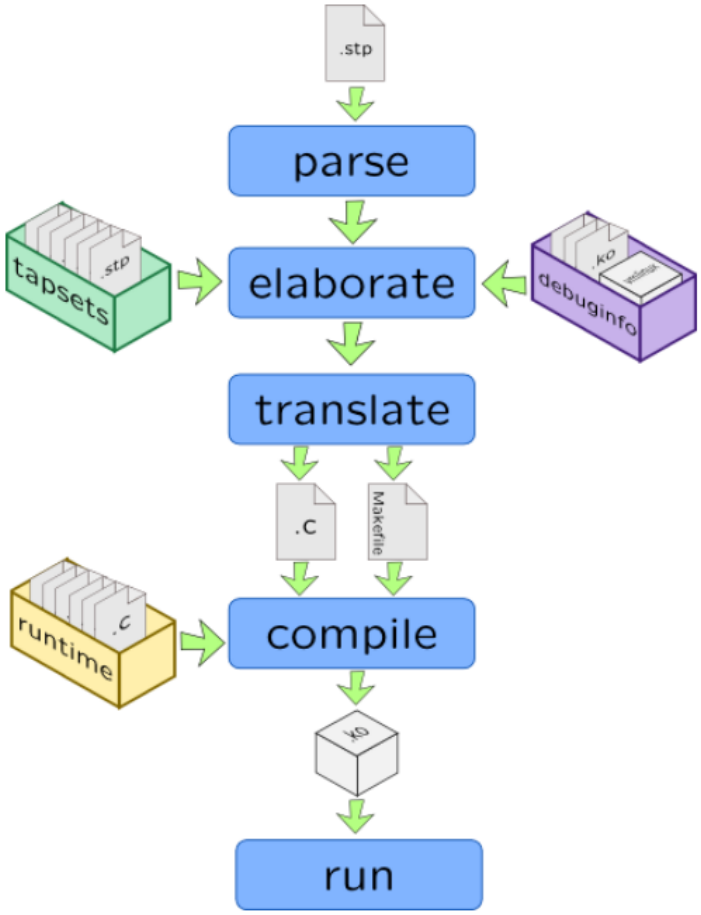
Some "guru mode" scripts may also have embedded C, which may run with the `-g` command-line option. However, use of guru mode is discouraged, and each SystemTap release includes more probe points designed to remove the need for guru-mode scripts. Guru mode is required in order to permit scripts to *modify* state in the instrumented software, such as to apply some types of emergency security fixes.

As of SystemTap version 1.7, the software implements the new *stapsys* group and privilege level.[10]

- https://sourceware.org/systemtap/wiki
- https://sourceware.org/git/?p=systemtap.git;a=summary
- …

How it works

SystemTap doesn't have VM in-kernel (unlike DTrace and KTap), instead it generates kernel module source written in C than builds it, so you will also need a compiler toolchain (`make`, `gcc` and `ld`). Compilation takes five phases: *parse*, *elaborate* in which tapsets and debuginfo is linked with script, *translate* in which C code is generated, *compile* and *run*:



Source: https://myaut.github.io/dtrace-stap-book/tools/systemtap.html

Examples

The following script shows all applications setting TCP socket options on the system, what options are being set, and whether the option is set successfully or not.

```
# Show sockets setting options

# Return enabled or disabled based on value of optval
function getstatus(optval)
{
    if ( optval == 1 )
        return "enabling"
    else
        return "disabling"
}

probe begin
{
    print ("\nChecking for apps setting socket options\n")
}

    # Set a socket option
    probe tcp.setsockopt
    {
        status = getstatus(user_int($optval))
        printf ("  App '%s' (PID %d) is %s socket option %s... ", execname(), pid(), status, optstr)
    }

        # Check setting the socket option worked
        probe tcp.setsockopt.return
        {
            if ( ret == 0 )
                printf ("success")
            else
                printf ("failed")
            printf ("\n")
        }

probe end
{
    print ("\nClosing down\n")
}
```

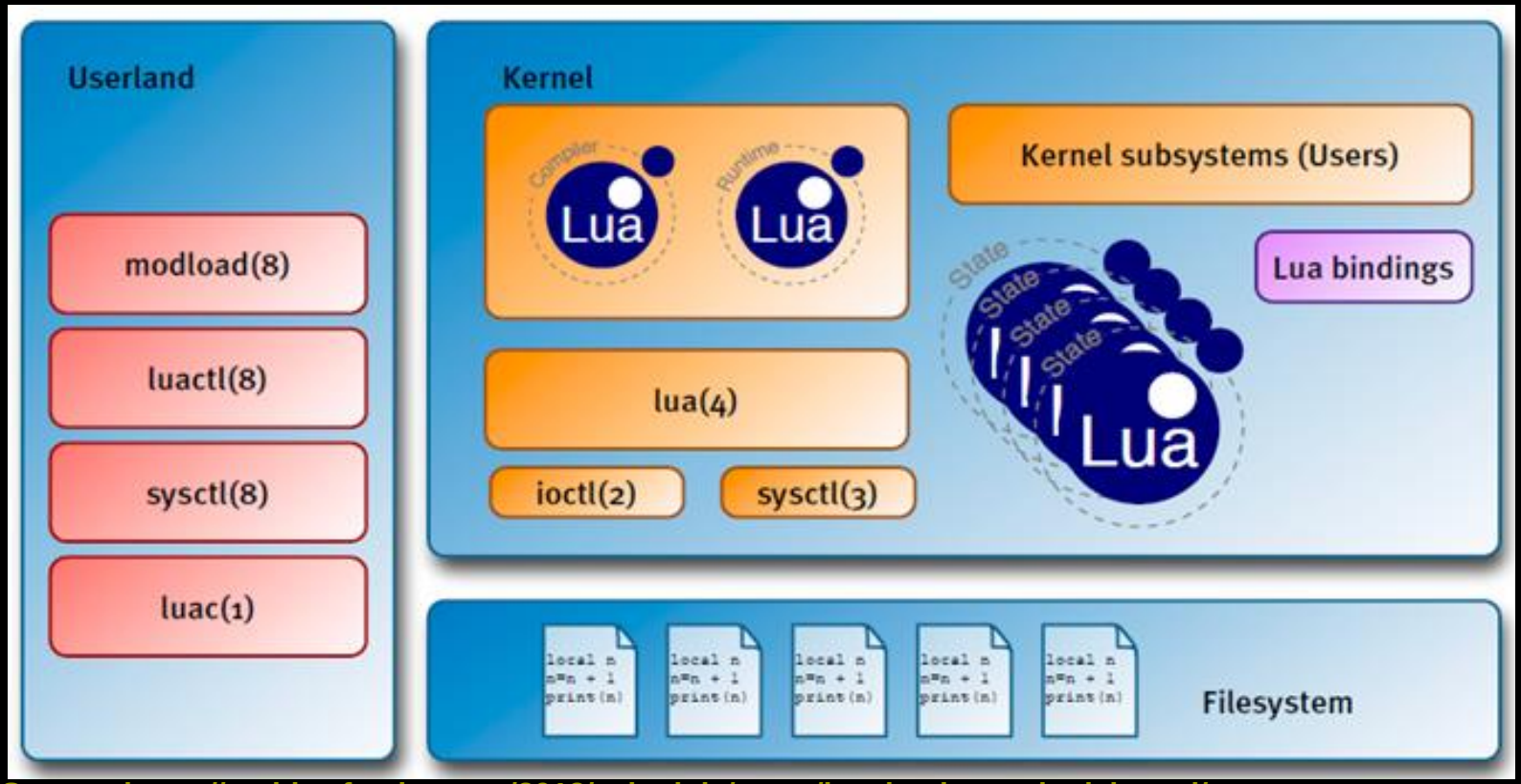
Source: https://en.wikipedia.org/wiki/SystemTap

- https://sourceware.org/systemtap/examples/
- https://sourceware.org/systemtap/wiki/WarStories
- https://sourceware.org/systemtap//SystemTap_Beginners_Guide.pdf
- ...

1.3.2 Lua-based in-kernel VM

1.3.2.1 NetBSD

- NetBSD Kernel scripting with Lua
  be part of NetBSD 6 (Userland)
  be part of NetBSD 7 (Kernel)



Source: https://archive.fosdem.org/2013/schedule/event/lua_in_the_netbsd_kernel/

1.3.2.2 Linux

- https://lwn.net/Articles/830154/ //Lua in the kernel?
- https://github.com/luainkernel

lunatik

- https://github.com/cujoai/lunatik
- a port of the Lua interpreter to the Linux kernel

lunatik-ng

- https://github.com/lunatik-ng/lunatik-ng
-

This repository contains the ongoing effort of porting the Lunatik Lua engine to current Linux kernels. There are a few differences between the original lunatik and lunatik-ng:

- Lunatik-ng works on x86_64
- It is memory-leak free
- It can be built as loadable modules
- A few interfaces to the kernel are provided by default:
  - `buffer` for allocating memory regions in kernel space
  - `crypto` which provides bindings to the SHA1 implementation in the kernel (a more advanced interface to the kernel which allows selection of the cipher is in the works) and the random number generator.
  - `printk` as a direct binding to the kernels `printk`
  - `type` and `gc_count` as bindings to parts of the default Lua library

KTap

- https://github.com/ktap/ktap

ktap is a new scripting dynamic tracing tool for Linux, it uses a scripting language and lets users trace the Linux kernel dynamically. ktap is designed to give operational insights with interoperability that allows users to tune, troubleshoot and extend the kernel and applications. It's similar to Linux Systemtap and Solaris Dtrace.

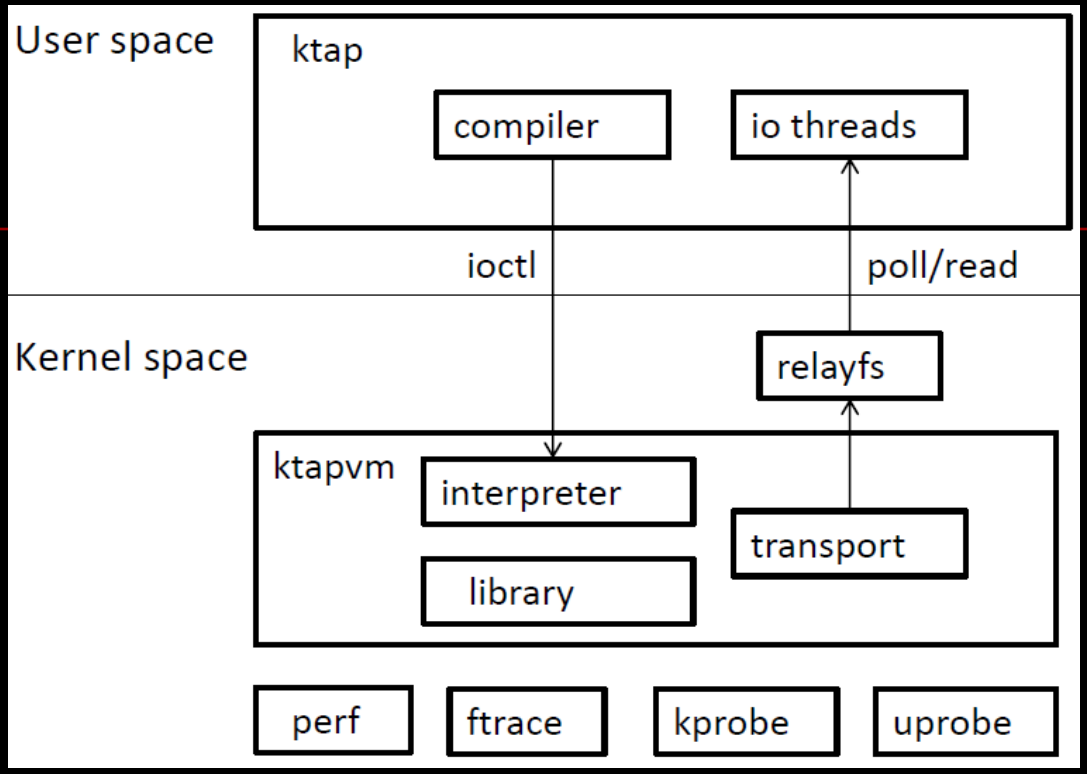
ktap has different design principles from Linux mainstream dynamic tracing language in that it's based on bytecode, so it doesn't depend upon GCC, doesn't require compiling kernel module for each script, safe to use in production environment, fulfilling the embedded ecosystem's tracing needs.

```
* ktap code is based on luajit(compiler & bytecode), so carry luajit
  copyright notices in below.

LuaJIT -- a Just-In-Time Compiler for Lua. http://luajit.org/
```

- History

https://lwn.net/Articles/551314/ //Ktap—yet another kernel tracer
https://lwn.net/Articles/572788/ //Ktap almost gets into 3.13
https://lwn.net/Articles/595565/ //Ktap or BPF?
https://lwn.net/Articles/595581/ //ktap and ebpf integration

Architecture



Source: "Ktap--A New Scripting Dynamic Tracing Tool For Linux", Wei Zhang(Huawei)," LinuxCon Japan 2013.

2) cBPF (classic Berkeley Packet Filter)

- https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

The **Berkeley Packet Filter** (**BPF**) is a technology used in certain computer operating systems for programs that need to, among other things, analyze network traffic. It provides a raw interface to data link layers, permitting raw link-layer packets to be sent and received.[1] In addition, if the driver for the network interface supports promiscuous mode, it allows the interface to be put into that mode so that all packets on the network can be received, even those destined to other hosts.

BPF supports filtering packets, allowing a userspace process to supply a filter program that specifies which packets it wants to receive. For example, a tcpdump process may want to receive only packets that initiate a TCP connection. BPF returns only packets that pass the filter that the process supplies. This avoids copying unwanted packets from the operating system kernel to the process, greatly improving performance. The filter program is in the form of instructions for a virtual machine, which are interpreted, or compiled into machine code by a just-in-time (JIT) mechanism and executed, in the kernel.

BPF is sometimes used to refer to just the filtering mechanism, rather than to the entire interface. Some systems, such as Linux and Tru64 UNIX, provide a raw interface to the data link layer other than the BPF raw interface but use the BPF filtering mechanisms for that raw interface. The BPF filtering mechanism is available on most Unix-like operating systems.

Berkeley Packet Filter

Developer(s)	Steven McCanne, Van Jacobson
Initial release	December 19, 1992; 29 years ago
Operating system	Unix-like (FreeBSD, OpenBSD, NetBSD, DragonFly BSD, macOS, Oracle Solaris 11 and later, AIX, Tru64, Linux, Orbis), Windows

Raw data-link interface [edit]

BPF provides pseudo-devices that can be bound to a network interface; reads from the device will read buffers full of packets received on the network interface, and writes to the device will inject packets on the network interface.

In 2007, Robert Watson and Christian Peron added zero-copy buffer extensions to the BPF implementation in the FreeBSD operating system,[3] allowing kernel packet capture in the device driver interrupt handler to write directly to user process memory in order to avoid the requirement for two copies for all packet data received via the BPF device. While one copy remains in the receipt path for user processes, this preserves the independence of different BPF device consumers, as well as allowing the packing of headers into the BPF buffer rather than copying complete packet data.[4]

Filtering [edit]

BPF's filtering capabilities are implemented as an interpreter for a machine language for the BPF virtual machine, a 32-bit machine with fixed-length instructions, one accumulator, and one index register. Programs in that language can fetch data from the packet, perform arithmetic operations on data from the packet, and compare the results against constants or against data in the packet or test bits in the results, accepting or rejecting the packet based on the results of those tests.

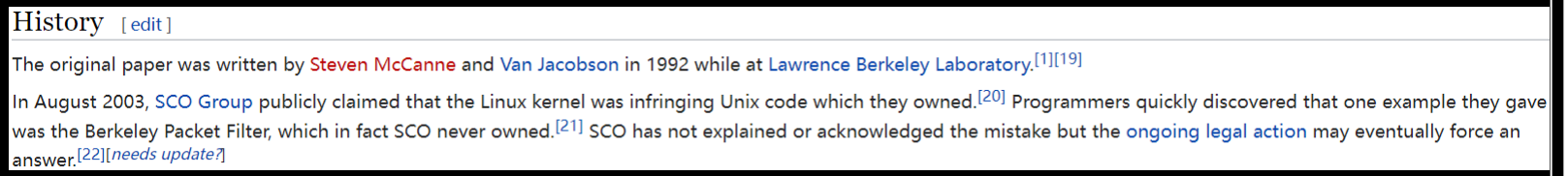
BPF is often extended by "overloading" the load (ld) and store (str) instructions.

Traditional Unix-like BPF implementations can be used in userspace, despite being written for kernel-space. This is accomplished using preprocessor conditions.

History

- Before BPF, each OS (Sun, DEC, SGI etc) had its own packet filtering API
- In 1993: Steven McCanne & Van Jacobsen released a paper titled the *BSD Packet Filter (BPF)*
- Implemented as “Linux Socket Filter” in kernel 2.2
- While maintaining the BPF language (for describing filters), uses a different internal architecture

Source: https://www.slideshare.net/MichaelKehoe3/ebpf-basics-149201150



History [edit]

The original paper was written by Steven McCanne and Van Jacobson in 1992 while at Lawrence Berkeley Laboratory.[1][19]

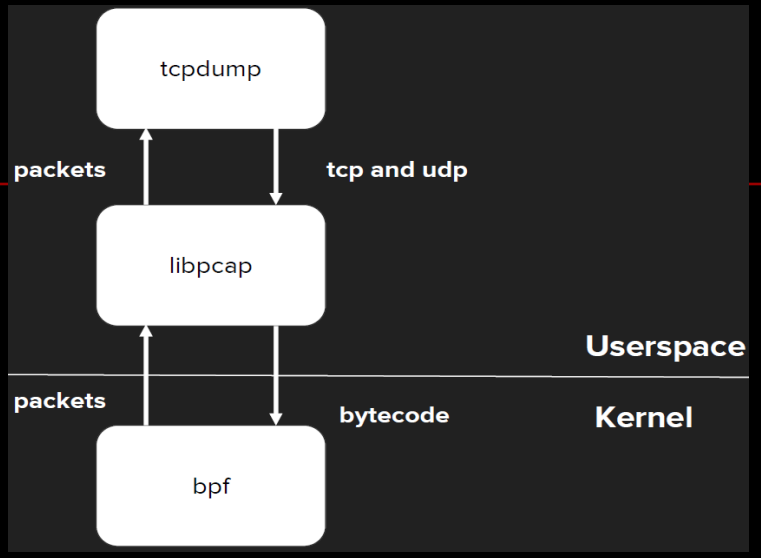
In August 2003, SCO Group publicly claimed that the Linux kernel was infringing Unix code which they owned.[20] Programmers quickly discovered that one example they gave was the Berkeley Packet Filter, which in fact SCO never owned.[21] SCO has not explained or acknowledged the mistake but the ongoing legal action may eventually force an answer.[22][*needs update?*]

Source: https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

...

Design & Implementation

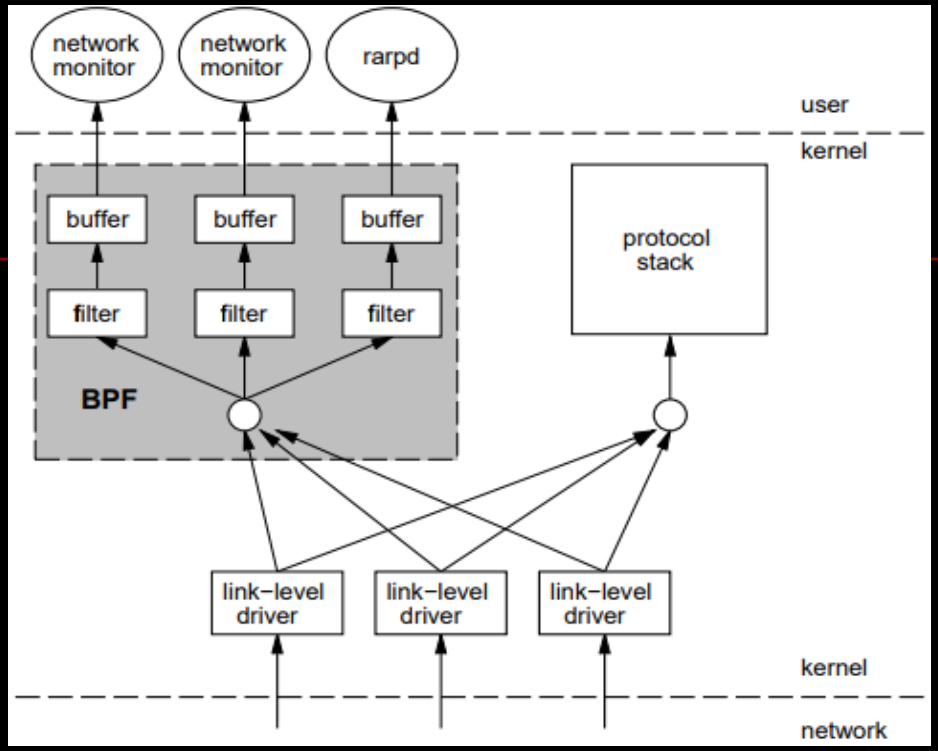
- Network packet filtering, Seccomp
- Filter Expressions → Bytecode → Interpret
- Small, in-kernel VM, Register based, switch dispatch interpreter, few instructions
- BPF uses a simple, non-shared buffer model made possible by today's larger address space



Source: https://www.slideshare.net/MichaelKehoe3/ebpf-basics-149201150

- Bytecode, register based VM, with a limited instruction set
- Runs in-kernel, designed for fast packet filtering
- 32-bit instructions (LOAD, STORE, ALU, BRANCH, RETURN)
- 2, 32-bit registers (A, X), hidden frame pointer

Source: https://www.slideshare.net/RayJenkins1/understanding-ebpf-in-a-hurry-149197981



Source: https://www.tcpdump.org/papers/bpf-usenix93.pdf

- Utilizes sockets for passing/receiving packets to/from the kernel-space
- Filters are attached with the setsockopt(2) system call
- Create a special-purpose socket (i.e., PF_PACKET) 2
- Attach a BPF program to the socket using the setsockopt(2) system call
- Set the network interface to promiscuous mode with ioctl(2) (optionally)
- Set various BPF parameters, (e.g. buffer size, attach some BPF filters) This is done using the ioctl(2) system call
- Read packets from the kernel, or send raw packets, by reading/writing to the corresponding file descriptor of /dev/bpf using read(2)/write(2) system calls
- Read packets from the kernel, or send raw packets, by reading/writing to the file descriptor of the socket using recvfrom(2)/sendto(2) system calls

Source: https://www.slideshare.net/MichaelKehoe3/ebpf-basics-149201150

Bytecode/Assembly

opcode:16	jt:8	jf:8
k:32		

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					

#k	the literal value stored in *k*
#len	the length of the packet
M[k]	the word at offset *k* in the scratch memory store
[k]	the byte, halfword, or word at byte offset *k* in the packet
[x+k]	the byte, halfword, or word at offset *x*+*k* in the packet
L	an offset from the current instruction to L
#k, Lt, Lf	the offset to Lt if the predicate is true, otherwise the offset to Lf
x	the index register
4*([k]&0xf)	four times the value of the low four bits of the byte at offset *k* in the packet

Source: https://www.tcpdump.org/papers/bpf-usenix93.pdf

an example:

```
# tcpdump host 127.0.0.1 and port 22 -d
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 18
(002) ld       [26]
(003) jeq      #0x7f000001      jt 6    jf 4
(004) ld       [30]
(005) jeq      #0x7f000001      jt 6    jf 18
(006) ldb      [23]
(007) jeq      #0x84            jt 10   jf 8
(008) jeq      #0x6             jt 10   jf 9
(009) jeq      #0x11            jt 10   jf 18
(010) ldh      [20]
(011) jset     #0x1fff          jt 18   jf 12
(012) ldxb     4*([14]&0xf)
(013) ldh      [x + 14]
[...]
```

Optimizes packet filter performance

2 x 32-bit registers & scratch memory

User-defined bytecode executed by an in-kernel sandboxed virtual machine

Steven McCanne and Van Jacobson, 1993

Source: https://www.slideshare.net/brendangregg/kernel-recipes-2017-performance-analysis-with-bpf

https://blog.cloudflare.com/bpf-the-forgotten-bytecode/

https://lwn.net/Articles/437981/ //A JIT for packet filters

…

3) eBPF(extended Berkeley Packet Filter)

3.1 Overview

- https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

Extensions and optimizations [edit]

Some projects use BPF instruction sets or execution techniques different from the originals.

Some platforms, including FreeBSD, NetBSD, and WinPcap, use a just-in-time (JIT) compiler to convert BPF instructions into native code in order to improve performance. Linux includes a BPF JIT compiler which is disabled by default.

Kernel-mode interpreters for that same virtual machine language are used in raw data link layer mechanisms in other operating systems, such as Tru64 Unix, and for socket filters in the Linux kernel and in the WinPcap and Npcap packet capture mechanism.

Since version 3.18, the Linux kernel includes an extended BPF virtual machine with ten 64-bit registers, termed **extended BPF** (**eBPF**). It can be used for non-networking purposes, such as for attaching eBPF programs to various tracepoints.[5][6][7] Since kernel version 3.19, eBPF filters can be attached to sockets,[8][9] and, since kernel version 4.1, to traffic control classifiers for the ingress and egress networking data path.[10][11] The original and obsolete version has been retroactively renamed to **classic BPF** (**cBPF**). Nowadays, the Linux kernel runs eBPF only and loaded cBPF bytecode is transparently translated into an eBPF representation in the kernel before program execution.[12] All bytecode is verified before running to prevent denial-of-service attacks. Until Linux 5.3, the verifier prohibited the use of loops.[13]

A user-mode interpreter for BPF is provided with the libpcap/WinPcap/Npcap implementation of the pcap API, so that, when capturing packets on systems without kernel-mode support for that filtering mechanism, packets can be filtered in user mode; code using the pcap API will work on both types of systems, although, on systems where the filtering is done in user mode, all packets, including those that will be filtered out, are copied from the kernel to user space. That interpreter can also be used when reading a file containing packets captured using pcap.

- Since Linux Kernel v3.15 and ongoing
- Aims at being a universal in-kernel virtual machine
- A simple way to extend the functionality of Kernel at runtime
- "DTrace for Linux“
- $KERNEL_SRC/Documentation/networking/filter.txt

Kernel configuration

```
[mydev@fedora boot]$ uname -a
Linux fedora 5.19.9-200.fc36.aarch64 #1 SMP PREEMPT_DYNAMIC Thu Sep 15 09:32:25 UTC 2022 aarch64 aarch64 aarch64 GNU/Linux
[mydev@fedora boot]$
[mydev@fedora boot]$ cat ./config-5.19.9-200.fc36.aarch64 |grep -i bpf
CONFIG_BPF=y
CONFIG_HAVE_EBPF_JIT=y
CONFIG_ARCH_WANT_DEFAULT_BPF_JIT=y
# BPF subsystem
CONFIG_BPF_SYSCALL=y
CONFIG_BPF_JIT=y
CONFIG_BPF_JIT_ALWAYS_ON=y
CONFIG_BPF_JIT_DEFAULT_ON=y
CONFIG_BPF_UNPRIV_DEFAULT_OFF=y
CONFIG_BPF_PRELOAD=y
CONFIG_BPF_PRELOAD_UMD=m
CONFIG_BPF_LSM=y
# end of BPF subsystem
CONFIG_CGROUP_BPF=y
CONFIG_IPV6_SEG6_BPF=y
CONFIG_NETFILTER_XT_MATCH_BPF=m
# CONFIG_BPFILTER is not set
CONFIG_NET_CLS_BPF=m
CONFIG_NET_ACT_BPF=m
CONFIG_BPF_STREAM_PARSER=y
CONFIG_LWTUNNEL_BPF=y
CONFIG_BPF_LIRC_MODE2=y
CONFIG_LSM="lockdown,yama,integrity,selinux,bpf,landlock"
CONFIG_BPF_EVENTS=y
# CONFIG_BPF_KPROBE_OVERRIDE is not set
CONFIG_TEST_BPF=m
[mydev@fedora boot]$
```

3.1.2 History

- https:///lwn.net/Articles/740157/ //A thorough introduction to eBPF

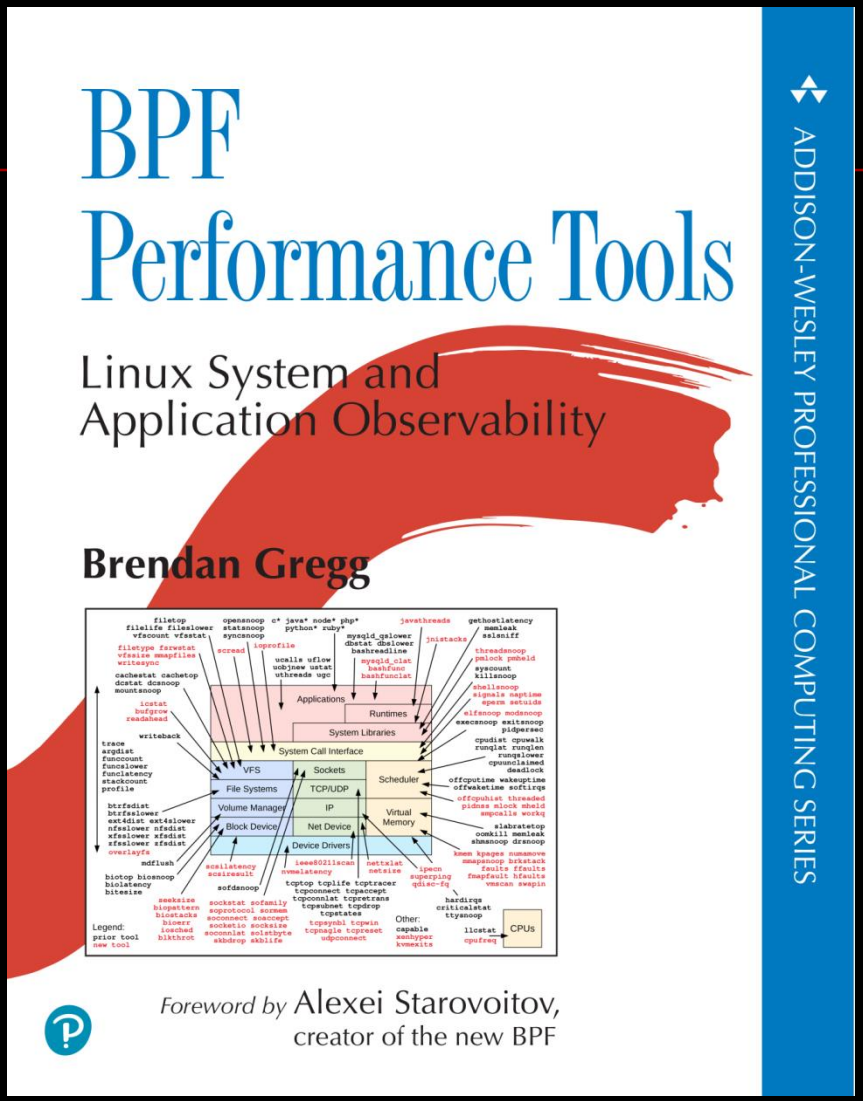
BPF Features by Linux Kernel Version

- https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md
- https://kernelnewbies.org/Linux_5.19

- BPF
  - (FEATURED) Add support for dynamic pointers. A dynamic pointer (`struct bpf_dynptr`) is a pointer that stores extra metadata alongside the address it points to. This abstraction is useful in bpf given that every memory access in a bpf program must be safe. The verifier and bpf helper functions can use the metadata to enforce safety guarantees for things such as dynamically sized strings and kernel heap allocations. There are several uses cases for dynamic pointers in bpf programs. Some examples include: dynamically sized ringbuf reservations without extra memcpys, dynamic string parsing and memory comparisons, dynamic memory allocations that can be persisted in maps, and dynamic + ergonomic parsing of sk_buff and xdp_md packet data commit , commit , commit , commit , commit , commit
  - (FEATURED) Introduce typed pointer support in BPF maps commit , commit , commit , commit , commit , commit , commit , commit , commit , commit , commit , commit , commit , commit
  - Introduce access remote cpu elem support in BPF percpu map commit
  - Refine `kernel.unprivileged_bpf_disabled` behaviour commit
  - Bpf link iterator commit
  - Allow attach TRACING programs through LINK_CREATE command commit
  - Allow kfunc in tracing and syscall programs commit
  - Extend batch operations for map-in-map bpf-maps commit
  - Add source ip in bpf tunnel key commit , commit , commit
  - Speed up symbol resolving in kprobe multi link commit , commit , commit , commit
- libbpf
  - (FEATURED) Add support for User Statically-Defined Tracing (USDTs) commit , commit , commit , commit , commit , commit , commit , commit , commit , commit
  - name-based u[ret]probe attach commit , commit , commit , commit , commit
  - Allow to opt-out from BPF map creation commit , commit , commit , commit
  - Add target-less tracing SEC() definitions commit , commit , commit
  - Support opting out from autoloading BPF programs declaratively commit
- bpftool: add program & link type names commit , commit , commit
- bpftool: Use sysfs vmlinux when dumping BTF by ID commit

...

- https://brendangregg.com/bpf-performance-tools-book.html
  https://github.com/brendangregg/bpf-perf-tools-book



3.2 ISA

- https://github.com/dthaler/ebpf-docs/blob/update/isa/kernel.org/instruction-set.rst
- https://github.com/iovisor/bpf-docs/blob/master/eBPF.md
- https://docs.kernel.org/bpf/instruction-set.html

Instruction Set

- registers and calling convention

The eBPF calling convention is defined as:

- R0: return value from function calls, and exit value for eBPF programs
- R1 - R5: arguments for function calls
- R6 - R9: callee saved registers that function calls will preserve
- R10: read-only frame pointer to access stack

Registers R0 - R5 are scratch registers, meaning the BPF program needs to either spill them to the BPF stack or move them to callee saved registers if these arguments are to be reused across multiple function calls. Spilling means that the value in the register is moved to the BPF stack. The reverse operation of moving the variable from the BPF stack to the register is called filling. The reason for spilling/filling is due to the limited number of registers.

> Note
>
> *Linux implementation*: In the Linux kernel, the exit value for eBPF programs is passed as a 32 bit value.

Upon entering execution of an eBPF program, registers R1 - R5 initially can contain the input arguments for the program (similar to the argc/argv pair for a typical C program). The actual number of registers used, and their meaning, is defined by the program type; for example, a networking program might have an argument that includes network packet data and/or metadata.

> Note
>
> *Linux implementation*: In the Linux kernel, all program types only use R1 which contains the "context", which is typically a structure containing all the inputs needed.

After execution of an eBPF program, register R0 contains the exit code whose meaning is defined by the program type, except that an exit code of -1 means the program was gracefully aborted. That is, if a program is gracefully aborted for any reason, it means that no futher instructions are executed, and a value of -1 is returned in register R0 to the caller of the program.

Source: https://github.com/dthaler/ebpf-docs/blob/update/isa/kernel.org/instruction-set.rst

eBPF instruction (bytecode) format

eBPF has two instruction encodings:

- the basic instruction encoding, which uses 64 bits to encode an instruction
- the wide instruction encoding, which appends a second 64-bit immediate (i.e., constant) value after the basic instruction for a total of 128 bits.

The basic instruction encoding is as follows, where MSB and LSB mean the most significant bits and least significant bits, respectively:

32 bits (MSB)	16 bits	4 bits	4 bits	8 bits (LSB)
imm	offset	src	dst	opcode

imm
signed integer immediate value

offset
signed integer offset used with pointer arithmetic

src
the source register number (0-10), except where otherwise specified (64-bit immediate instructions reuse this field for other purposes)

dst
destination register number (0-10)

opcode
operation to perform

Note that most instructions do not use all of the fields. Unused fields must be set to zero.

As discussed below in 64-bit immediate instructions, some instructions use a 64-bit immediate value that is constructed as follows. The 64 bits following the basic instruction contain a pseudo instruction using the same format but with opcode, dst, src, and offset all set to zero, and imm containing the high 32 bits of the immediate value.

64 bits (MSB)	64 bits (LSB)
basic instruction	pseudo instruction

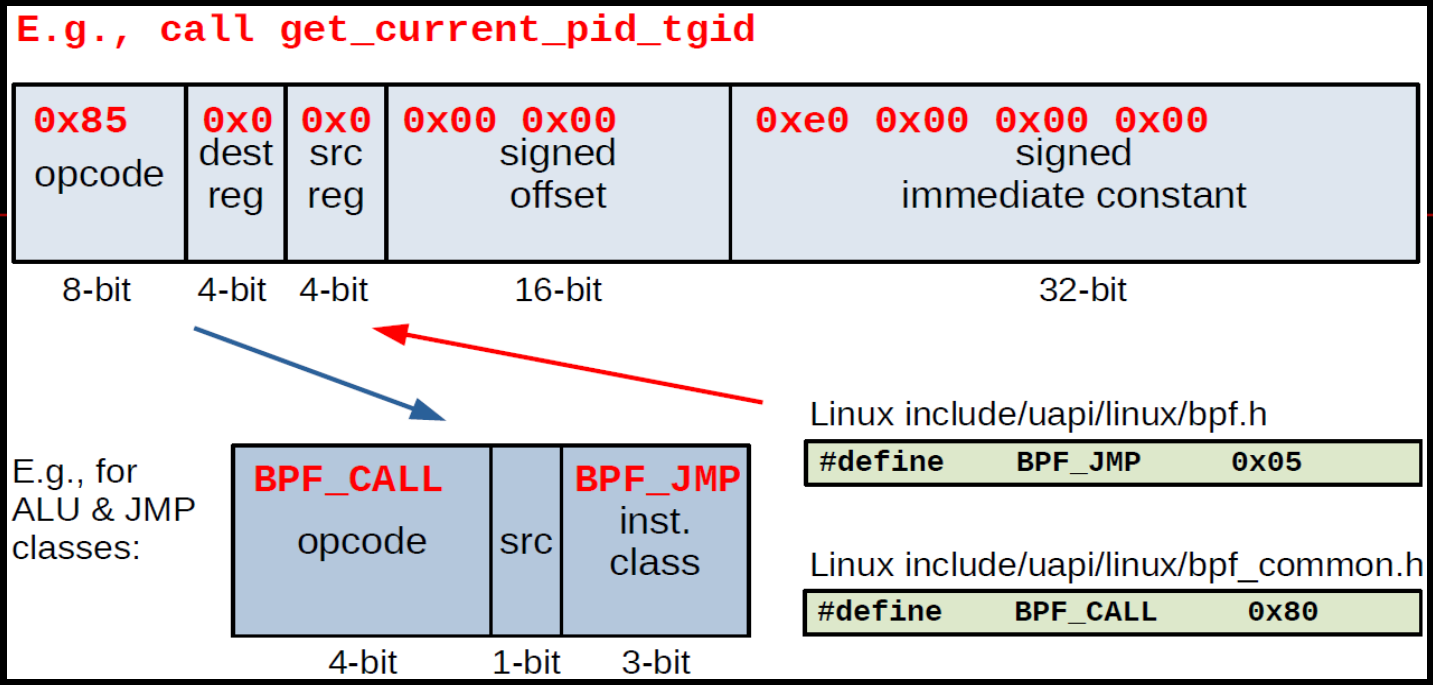
Thus the 64-bit immediate value is constructed as follows:

```
imm64 = imm + (next_imm << 32)
```

where 'next_imm' refers to the imm value of the pseudo instruction following the basic instruction.

Source: https://github.com/dthaler/ebpf-docs/blob/update/isa/kernel.org/instruction-set.rst

an example:



Source: https://www.brendangregg.com/Slides/LISA2021_BPF_Internals

Comparison

- Extended BPF (eBPF) modernized BPF

	Classic BPF	Extended BPF
Word size	32-bit	64-bit
Registers	2	10+1
Storage	16 slots	512 byte stack + infinite map storage
Events	packets	many event sources

Maintainers/creators: Alexei Starovoitov & Daniel Borkmann

Old BPF is now “Classic BPF,” and eBPF is usually just “BPF”

Source: https://www.brendangregg.com/Slides/LISA2021_BPF_Internals

- **https://docs.kernel.org/bpf/classic_vs_extended.html**

3.3 How it works

- bpf() system call

  http://www.man7.org/linux/man-pages/man2/bpf.2.html

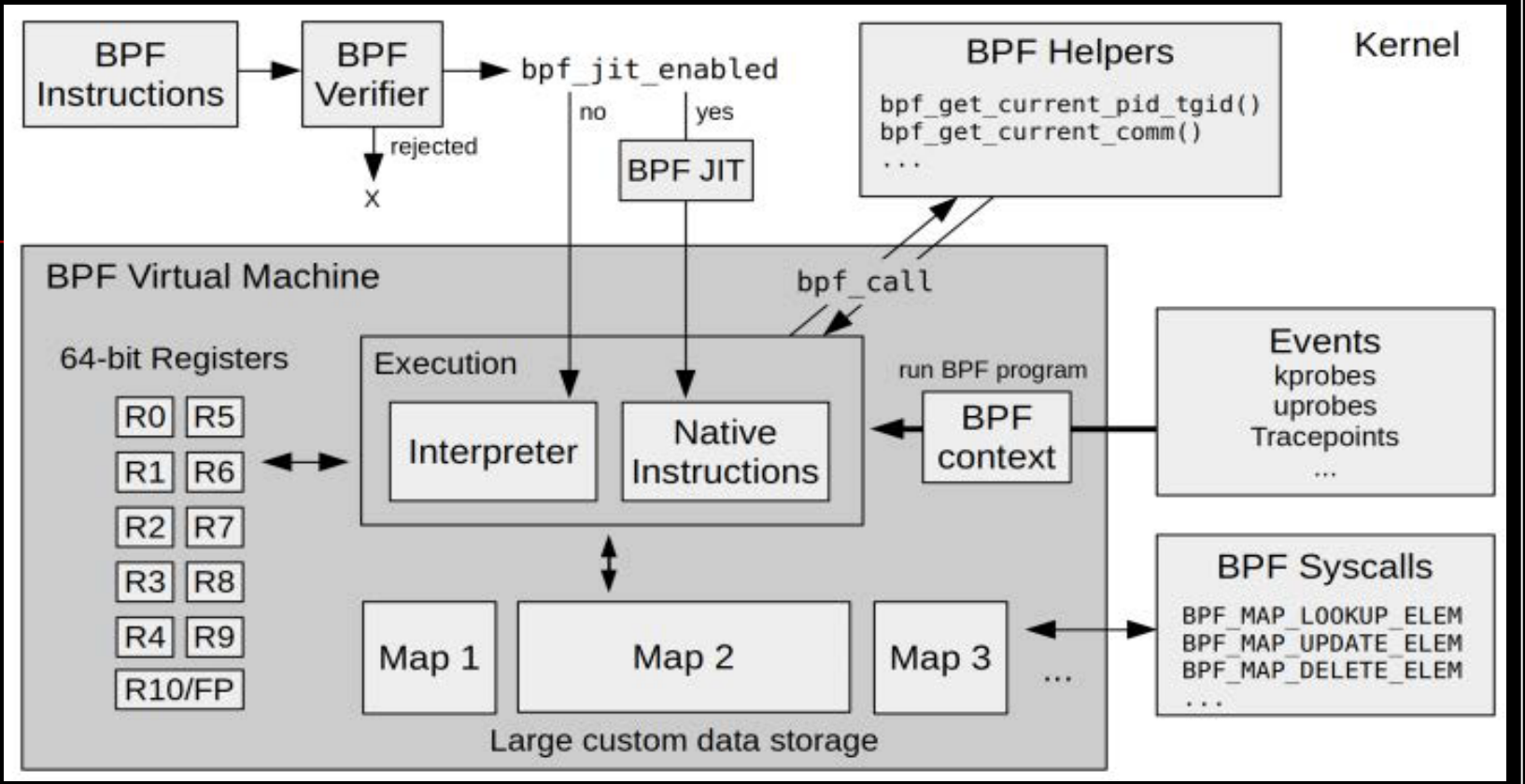
```
#include <linux/bpf.h>

int bpf(int cmd, union bpf_attr *attr, unsigned int size);
```

```
enum bpf_prog_type {
        BPF_PROG_TYPE_UNSPEC,
        BPF_PROG_TYPE_SOCKET_FILTER,
        BPF_PROG_TYPE_KPROBE,
        BPF_PROG_TYPE_SCHED_CLS,
        BPF_PROG_TYPE_SCHED_ACT,
        BPF_PROG_TYPE_TRACEPOINT,
        BPF_PROG_TYPE_XDP,
        BPF_PROG_TYPE_PERF_EVENT,
        BPF_PROG_TYPE_CGROUP_SKB,
        BPF_PROG_TYPE_CGROUP_SOCK,
        BPF_PROG_TYPE_LWT_IN,
        BPF_PROG_TYPE_LWT_OUT,
        BPF_PROG_TYPE_LWT_XMIT,
        BPF_PROG_TYPE_SOCK_OPS,
        BPF_PROG_TYPE_SK_SKB,
        BPF_PROG_TYPE_CGROUP_DEVICE,
        BPF_PROG_TYPE_SK_MSG,
        BPF_PROG_TYPE_RAW_TRACEPOINT,
        BPF_PROG_TYPE_CGROUP_SOCK_ADDR,
        BPF_PROG_TYPE_LWT_SEG6LOCAL,
        BPF_PROG_TYPE_LIRC_MODE2,
        BPF_PROG_TYPE_SK_REUSEPORT,
        BPF_PROG_TYPE_FLOW_DISSECTOR,
        BPF_PROG_TYPE_CGROUP_SYSCTL,
        BPF_PROG_TYPE_RAW_TRACEPOINT_WRITABLE,
        BPF_PROG_TYPE_CGROUP_SOCKOPT,
        BPF_PROG_TYPE_TRACING,
        BPF_PROG_TYPE_STRUCT_OPS,
        BPF_PROG_TYPE_EXT,
        BPF_PROG_TYPE_LSM,
        BPF_PROG_TYPE_SK_LOOKUP,
        BPF_PROG_TYPE_SYSCALL, /* a program that can execute syscalls */
};
```

The *bpf_attr* union consists of various anonymous structures that are used by different **bpf**() commands:

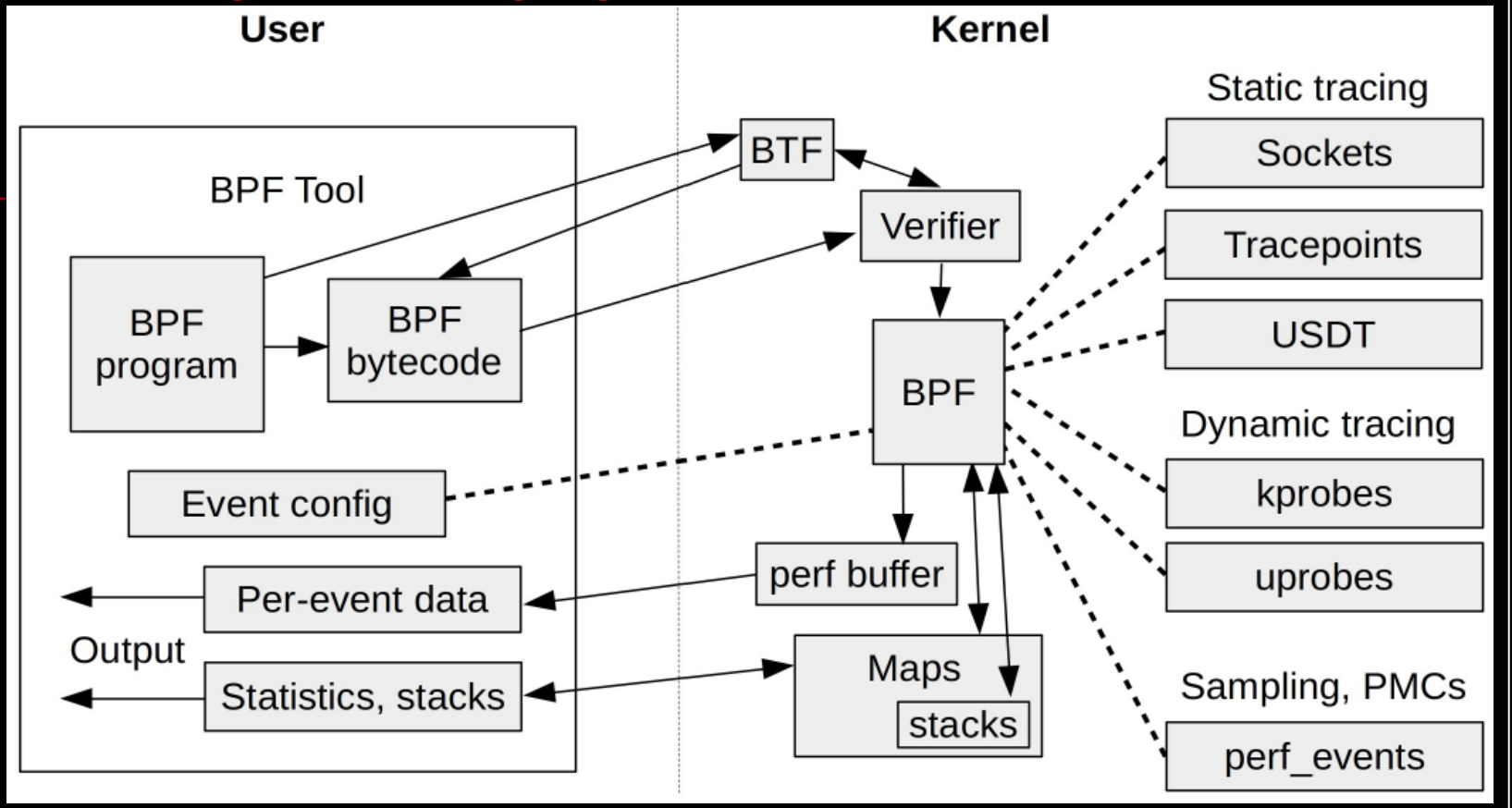
```
union bpf_attr {
    struct {    /* Used by BPF_MAP_CREATE */
        __u32         map_type;
        __u32         key_size;    /* size of key in bytes */
        __u32         value_size;  /* size of value in bytes */
        __u32         max_entries; /* maximum number of entries
                                      in a map */
    };

    struct {    /* Used by BPF_MAP_*_ELEM and BPF_MAP_GET_NEXT_KEY
                   commands */
        __u32         map_fd;
        __aligned_u64 key;
        union {
            __aligned_u64 value;
            __aligned_u64 next_key;
        };
        __u64         flags;
    };

    struct {    /* Used by BPF_PROG_LOAD */
        __u32         prog_type;
        __u32         insn_cnt;
        __aligned_u64 insns;      /* 'const struct bpf_insn *' */
        __aligned_u64 license;    /* 'const char *' */
        __u32         log_level;  /* verbosity level of verifier */
        __u32         log_size;   /* size of user buffer */
        __aligned_u64 log_buf;    /* user supplied 'char *'
                                     buffer */
        __u32         kern_version;
                                  /* checked when prog_type=kprobe
                                     (since Linux 4.1) */
    };
} __attribute__((aligned(8)));
```

```
enum bpf_map_type {
        BPF_MAP_TYPE_UNSPEC,
        BPF_MAP_TYPE_HASH,
        BPF_MAP_TYPE_ARRAY,
        BPF_MAP_TYPE_PROG_ARRAY,
        BPF_MAP_TYPE_PERF_EVENT_ARRAY,
        BPF_MAP_TYPE_PERCPU_HASH,
        BPF_MAP_TYPE_PERCPU_ARRAY,
        BPF_MAP_TYPE_STACK_TRACE,
        BPF_MAP_TYPE_CGROUP_ARRAY,
        BPF_MAP_TYPE_LRU_HASH,
        BPF_MAP_TYPE_LRU_PERCPU_HASH,
        BPF_MAP_TYPE_LPM_TRIE,
        BPF_MAP_TYPE_ARRAY_OF_MAPS,
        BPF_MAP_TYPE_HASH_OF_MAPS,
        BPF_MAP_TYPE_DEVMAP,
        BPF_MAP_TYPE_SOCKMAP,
        BPF_MAP_TYPE_CPUMAP,
        BPF_MAP_TYPE_XSKMAP,
        BPF_MAP_TYPE_SOCKHASH,
        BPF_MAP_TYPE_CGROUP_STORAGE,
        BPF_MAP_TYPE_REUSEPORT_SOCKARRAY,
        BPF_MAP_TYPE_PERCPU_CGROUP_STORAGE,
        BPF_MAP_TYPE_QUEUE,
        BPF_MAP_TYPE_STACK,
        BPF_MAP_TYPE_SK_STORAGE,
        BPF_MAP_TYPE_DEVMAP_HASH,
        BPF_MAP_TYPE_STRUCT_OPS,
        BPF_MAP_TYPE_RINGBUF,
        BPF_MAP_TYPE_INODE_STORAGE,
        BPF_MAP_TYPE_TASK_STORAGE,
        BPF_MAP_TYPE_BLOOM_FILTER,
};
```



Source: https://brendangregg.com/bpf-performance-tools-book.html

- **BPF tracing/observability high-level**

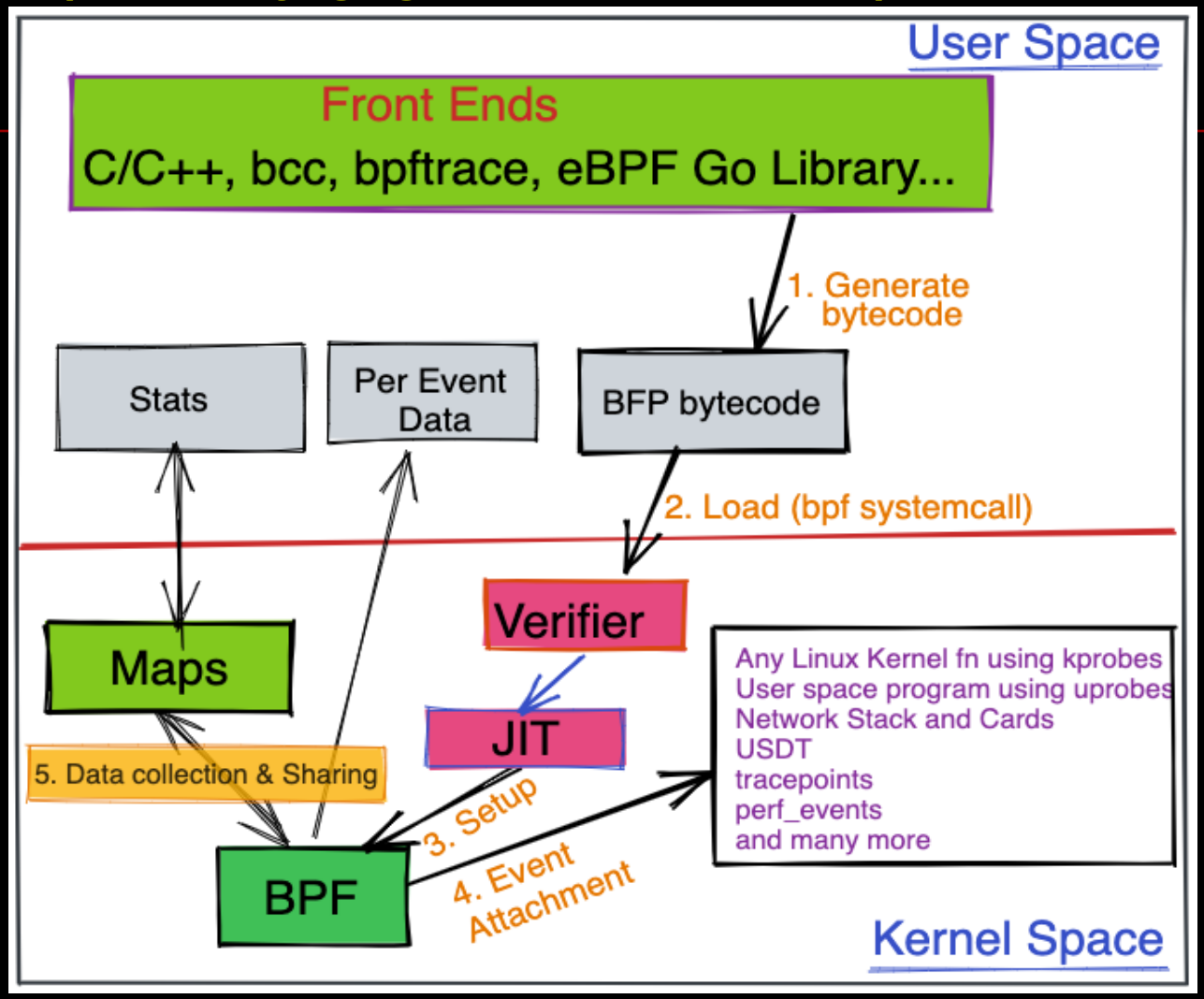


Source: https://brendangregg.com/bpf-performance-tools-book.html

- **https://docs.cilium.io/en/stable/bpf/**

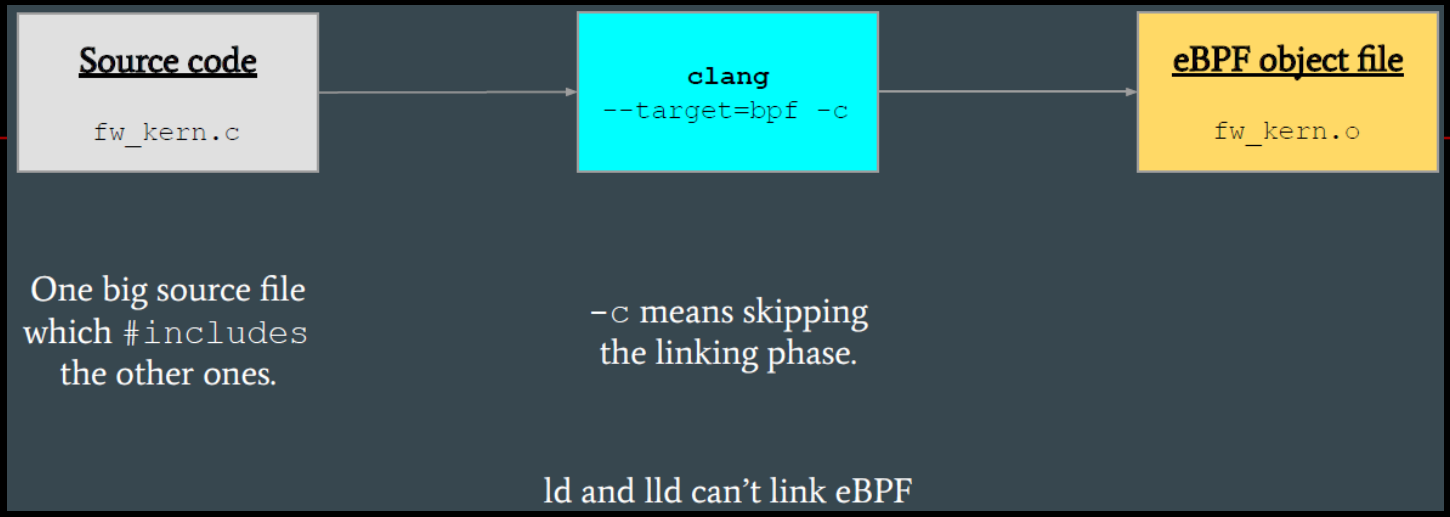
Workflow

- **https://cloudyuga.guru/hands_on_lab/ebpf-intro**



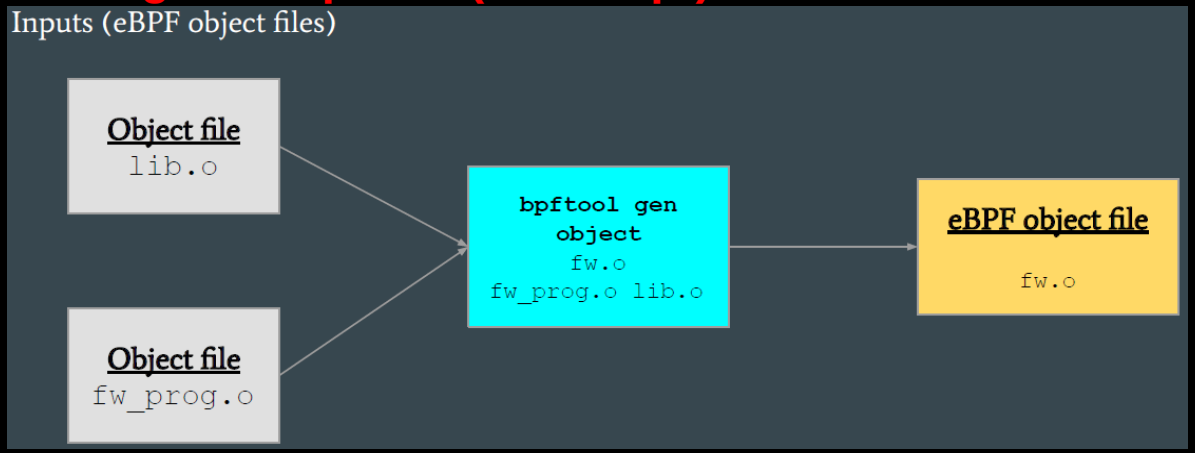
3.2.1 Compilation

- C -> eBPF compilation (the old, still the most popular way)



Source: "Rust in the Kernel (via eBPF)", Dave Tucker etc, LPC 2022.

- Linking with bpftool (via libbpf)



Source: "Rust in the Kernel (via eBPF)", Dave Tucker etc, LPC 2022.

3.4 XDP (eXpress Data Path)

- https://en.wikipedia.org/wiki/Express_Data_Path

XDP (**eXpress Data Path**) is an eBPF-based high-performance data path used to send and receive network packets at high rates by bypassing most of the operating system networking stack. It is merged in the Linux kernel since version 4.8.[2] This implementation is licensed under GPL. Large technology firms including Amazon, Google and Intel support its development. Microsoft released their free and open source implementation *XDP for Windows* in May 2022.[1] It is licensed under MIT License.[3]

Data path [edit]

The idea behind XDP is to add an early hook in the RX path of the kernel, and let a user supplied eBPF program decide the fate of the packet. The hook is placed in the network interface controller (NIC) driver just after the interrupt processing, and before any memory allocation needed by the network stack itself, because memory allocation can be an expensive operation. Due to this design, XDP can drop 26 million packets per second per core with commodity hardware.[4]

The eBPF program must pass a preverifier test[5] before being loaded, to avoid executing malicious code in kernel space. The preverifier checks that the program contains no out-of-bounds accesses, loops or global variables.

The program is allowed to edit the packet data and, after the eBPF program returns, an action code determines what to do with the packet:

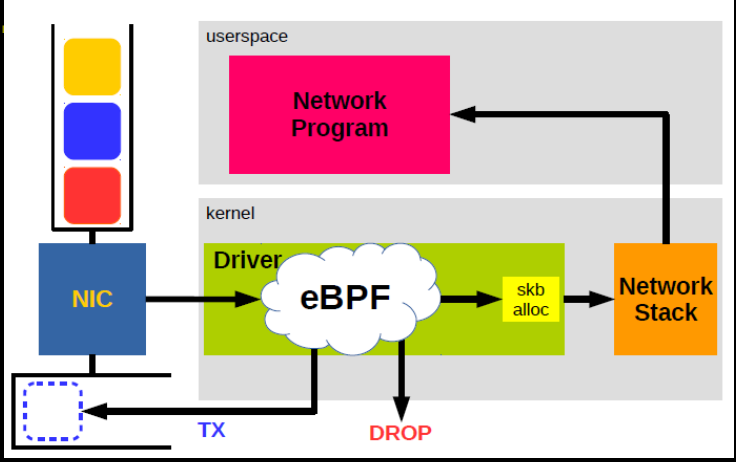
- `XDP_PASS` : let the packet continue through the network stack
- `XDP_DROP` : silently drop the packet
- `XDP_ABORTED` : drop the packet with trace point exception
- `XDP_TX` : bounce the packet back to the same NIC it arrived on
- `XDP_REDIRECT` : redirect the packet to another NIC or user space socket via the AF_XDP address family

XDP requires support in the NIC driver but, as not all drivers support it, it can fallback to a generic implementation, which performs the eBPF processing in the network stack, though with slower performance.[6]

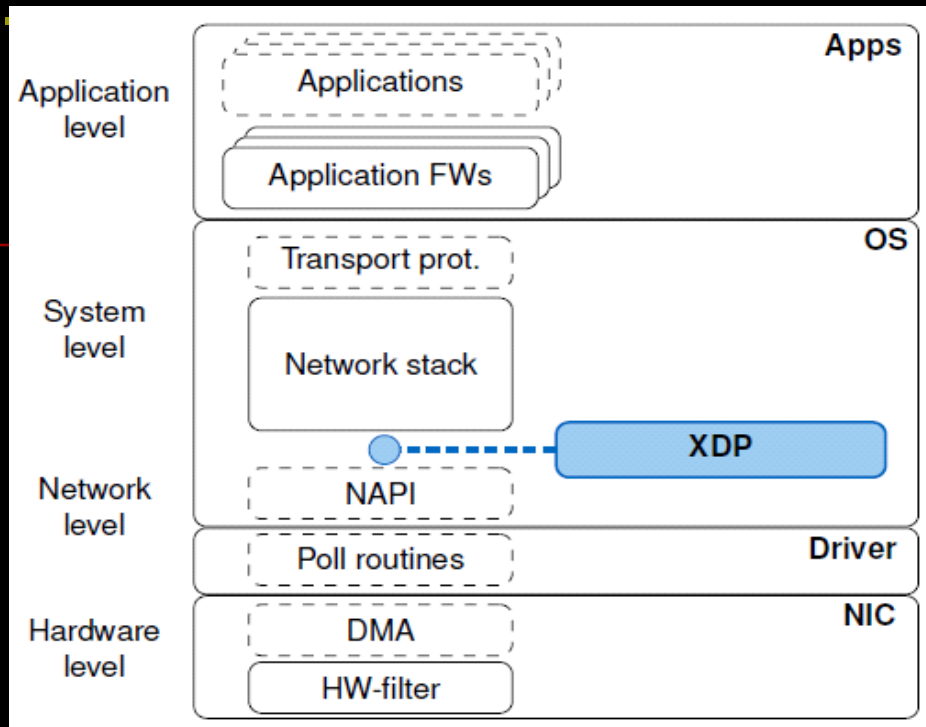
XDP has infrastructure to offload the eBPF program to a network interface controller which supports it, reducing the CPU load. At the time only Netronome cards supports it,[7] with Intel and Mellanox working on it.[8]

Microsoft is partnering with other companies and adding support for XDP in the MsQuic protocol.[1]

- https://www.iovisor.org/technology/xdp



Source: https://www.slideshare.net/lcplcp1/xdp-and-ebpfmaps



eXpress Data Path

- First line of defense
- Coarse but efficient filtering
- → Protection against DoS attacks

Source: https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/ITC30-Packet-Filtering-eBPF-XDP-slides.pdf

- https://lwn.net/Articles/708087/ //Debating the value of XDP
- https://www.seekret.io/blog/a-gentle-introduction-to-xdp/

Generic hook

- XDP is a further step in evolution and enables to run a specific flavor of BPF programs from the network driver with direct access to the packet's DMA buffer. This is, by definition, the earliest possible point in the software stack, where programs can be attached to in order to allow for a programmable, high performance packet processor in the Linux kernel networking data path.

Source: https://github.com/cilium/cilium

- **XDP Models**

Native XDP & Generic XDP

- XDP requires implementation in each driver
  - Need to choose XDP-supported driver
  - Not so handy
- Generic XDP allows you to use XDP on any driver (kernel 4.12)
  - XDP implemented in network stack
    - Convert skb to xdp buffer
  - Not as fast as native (non-generic) XDP
    - Need skb allocation at drivers
    - Packet buffer copy to meet XDP requirements
  - Good for functionality testing, etc.

Source: "Veth XDP--XDP for containers", Toshiaki Makita & William Tu, NetDev 2019.

Offloaded XDP

- **Offloaded XDP:** XDP program is loaded directly on the NIC. Thus it executes without using the node's CPU. This requires support from the network interface device.

Source: https://ltomasbo.wordpress.com/2022/01/10/openstack-with-bgp-accelerated-with-ebpf-xdp/

- ...

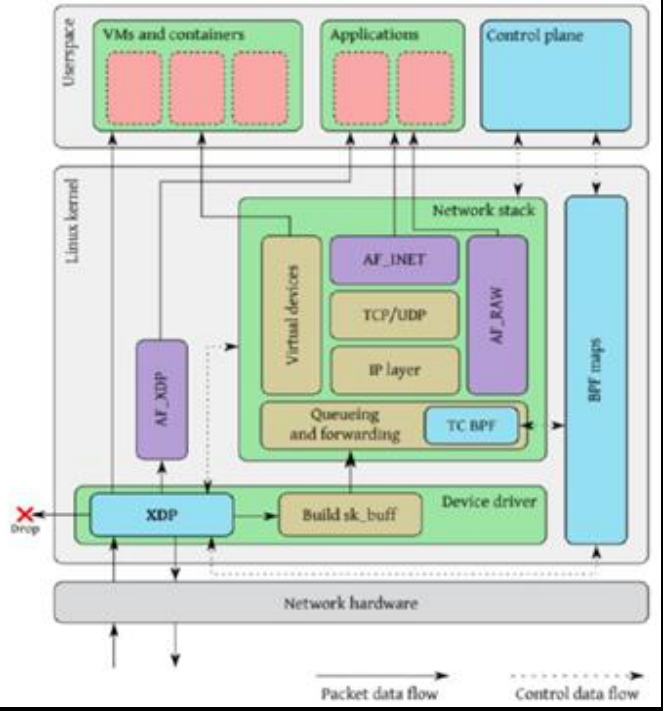
AF_XDP

- AF_XDP [edit]

  Along with XDP, a new address family entered in the Linux kernel starting 4.18.[9] AF_XDP, formerly known as AF_PACKETv4 (which was never included in the mainline kernel),[10] is a raw socket optimized for high performance packet processing and allows zero-copy between kernel and applications. As the socket can be used for both receiving and transmitting, it supports high performance network applications purely in user space.[11]

Source: https://en.wikipedia.org/wiki/Express_Data_Path

- https://www.kernel.org/doc/html/latest/networking/af_xdp.html
- https://lwn.net/Articles/750293/ //Introducing AF_XDP support
- https://lwn.net/Articles/750845/ //Accelerating networking with AF_XDP
- Overview
  - XDP's user space interface
  - Use XDP program to trigger Rx path for selected queue
  - Zero Copy from DMA buffers to user space with driver support
  - Copy mode for non-modified drivers
- Benefits
  - Performance boost
  - Support all Linux network devices

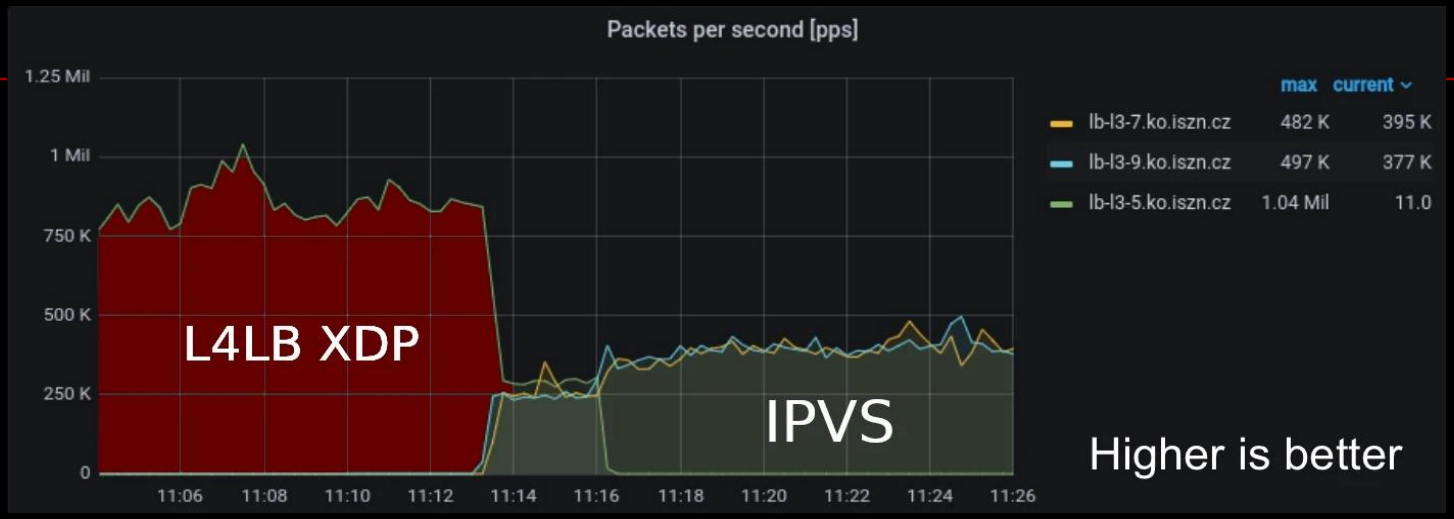


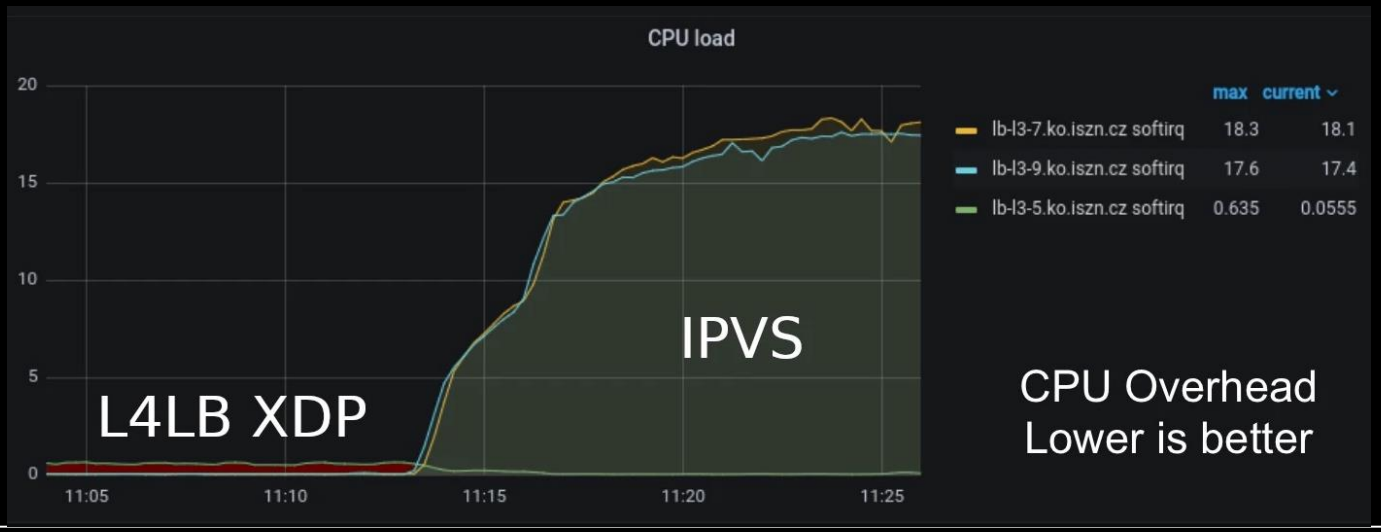
Source: "Integrating AF_XDP into DPDK", XIAO LONG YE, DPDK Summit China 2019.

Performance: Cilium Standalone Layer 4 Load Balancer XDP

- **https://cilium.io/blog/2022/04/12/cilium-standalone-L4LB-XDP/**

Production Traffic





3.5 uBPF

3.5.1 Overview

- userspace BPF
- https://stackoverflow.com/questions/65904948/why-is-having-an-userspace-version-of-ebpf-interesting:
  applied to Networking & Blockchain…
- You may refer to my previous talks and upcoming follow-ups.
  1. "GraalVM-based unified runtime for eBPF & Wasm" at GOTC 2021 (Shenzhen, Online);
  2. "Revisiting GraalVM-based unified runtime for eBPF & Wasm" at OpenInfra Days China 2021(Beijing, Online);

  …

II. eBPF in today's Linux Kernel

1) eBPF implementation in Kernel

- Working on a new detailed analysis of eBPF related code according to the upcoming Kernel 6.x;
- For a out-of-date version, you may refer to my previous talk "eBPF in Action" at LC3 2018(Beijing).

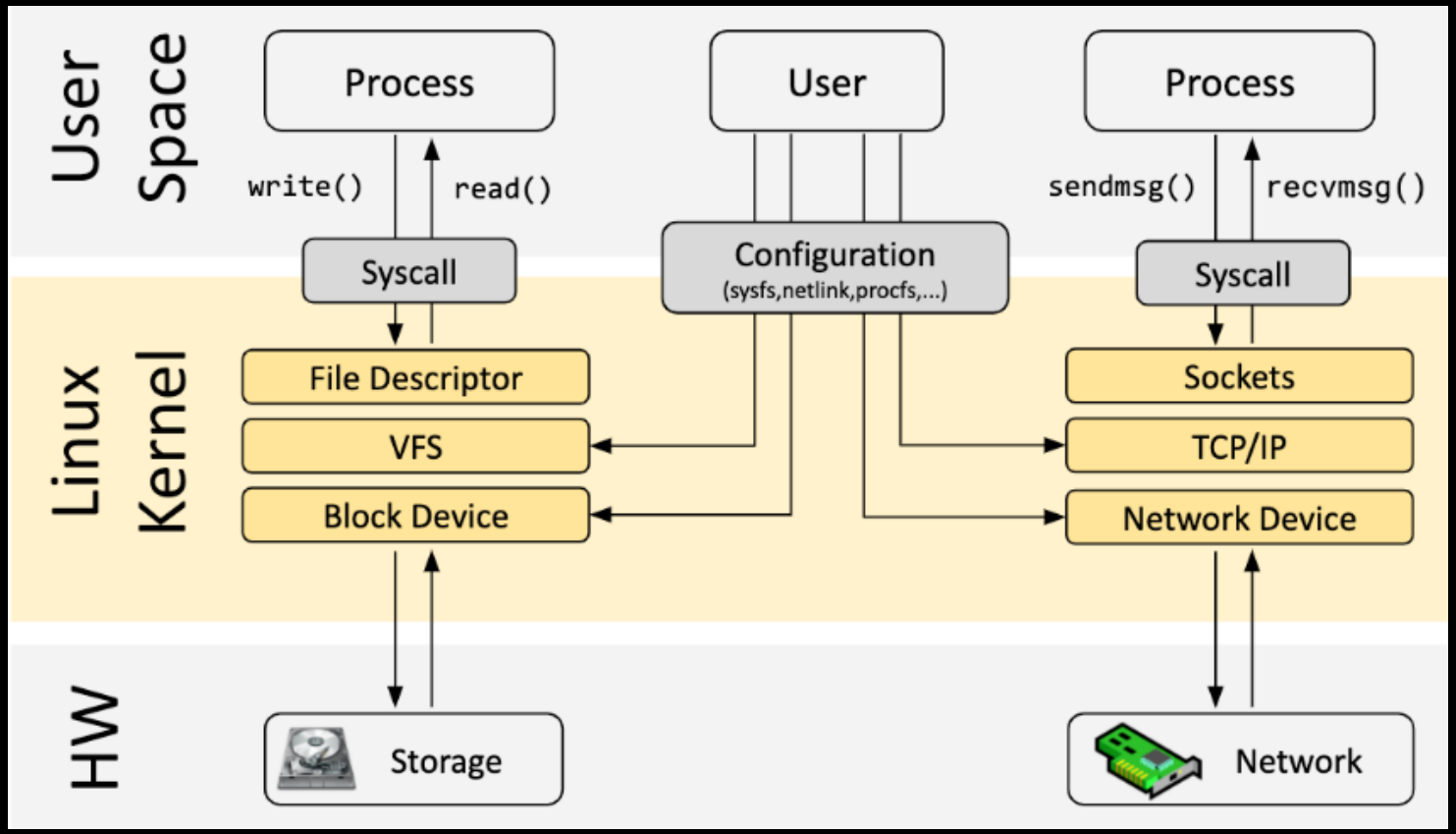
...

2) Kernel subsystems with eBPF

Overview

- **eBPF's impact on the Linux Kernel**

Now let's return to eBPF. In order to understand the programmability impact of eBPF on the Linux kernel, it helps to have a high-level understanding of the architecture of the Linux kernel and how it interacts with applications and the hardware.



Source: https://ebpf.io/what-is-ebpf

The main purpose of the Linux kernel is to abstract the hardware or virtual hardware and provide a consistent API (system calls) allowing for applications to run and share the resources. In order to achieve this, a wide set of subsystems and layers are maintained to distribute these responsibilities. Each subsystem typically allows for some level of configuration to account for different needs of users. If a desired behavior cannot be configured, a kernel change is required, historically, leaving two options:

Native Support	Kernel Module
1. Change kernel source code and convince the Linux kernel community that the change is required. 2. Wait several years for the new kernel version to become a commodity.	1. Write a kernel module 2. Fix it up regularly, as every kernel release may break it 3. Risk corrupting your Linux kernel due to lack of security boundaries

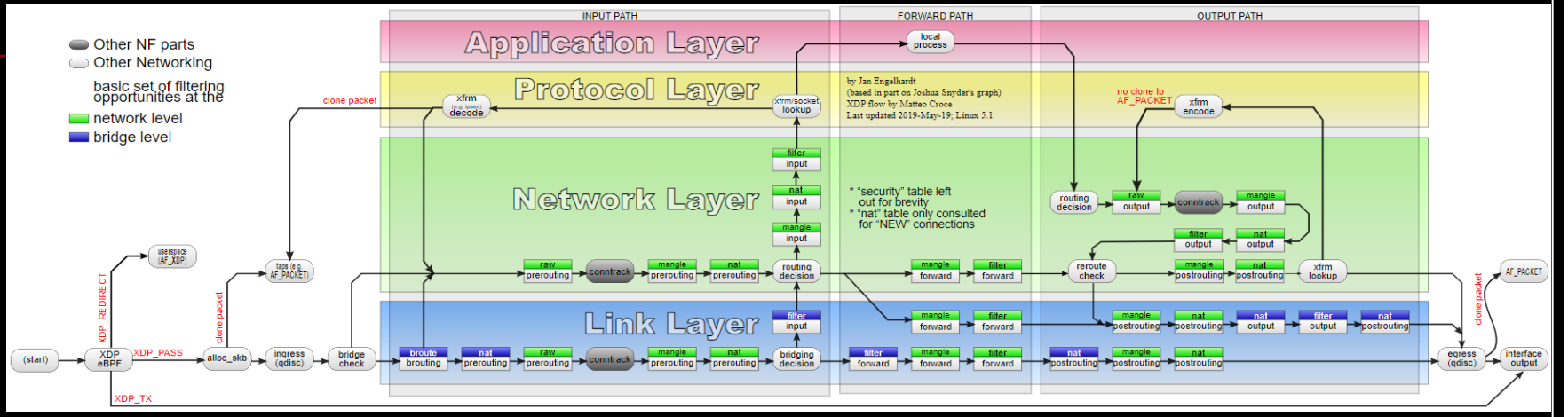
With eBPF, a new option is available that allows for reprogramming the behavior of the Linux kernel without requiring changes to kernel source code or loading a kernel module. In many ways, this is very similar to how JavaScript and other scripting languages unlocked the evolution of systems which had become difficult or expensive to change.

Source: https://ebpf.io/what-is-ebpf

2.1 Network

2.1.1 Linux Networking

- Packet flow in Netfilter and General Networking

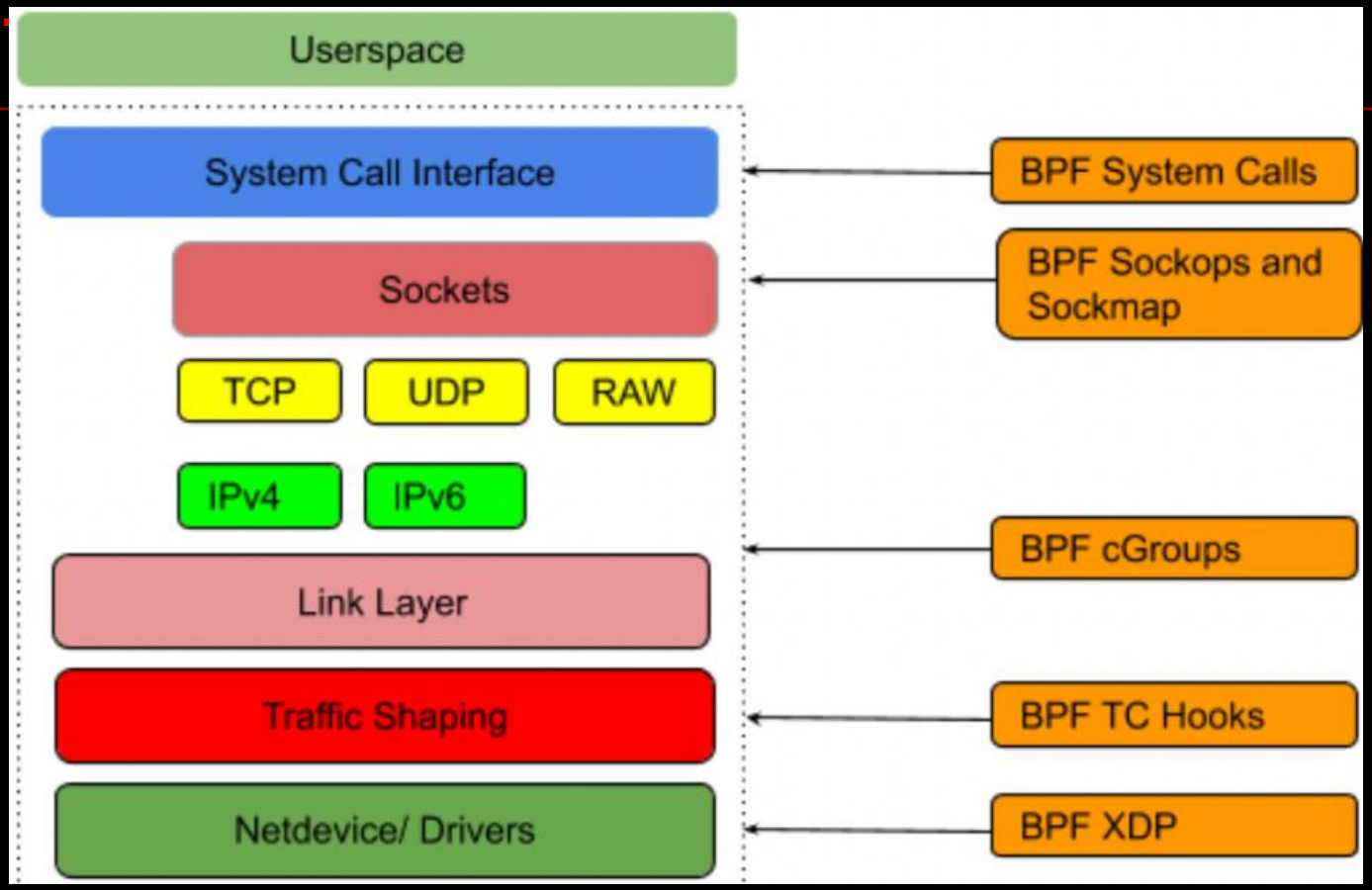


Source: https://upload.wikimedia.org/wikipedia/commons/3/37/Netfilter-packet-flow.svg

- …

2.1.2 eBPF kernel hooks

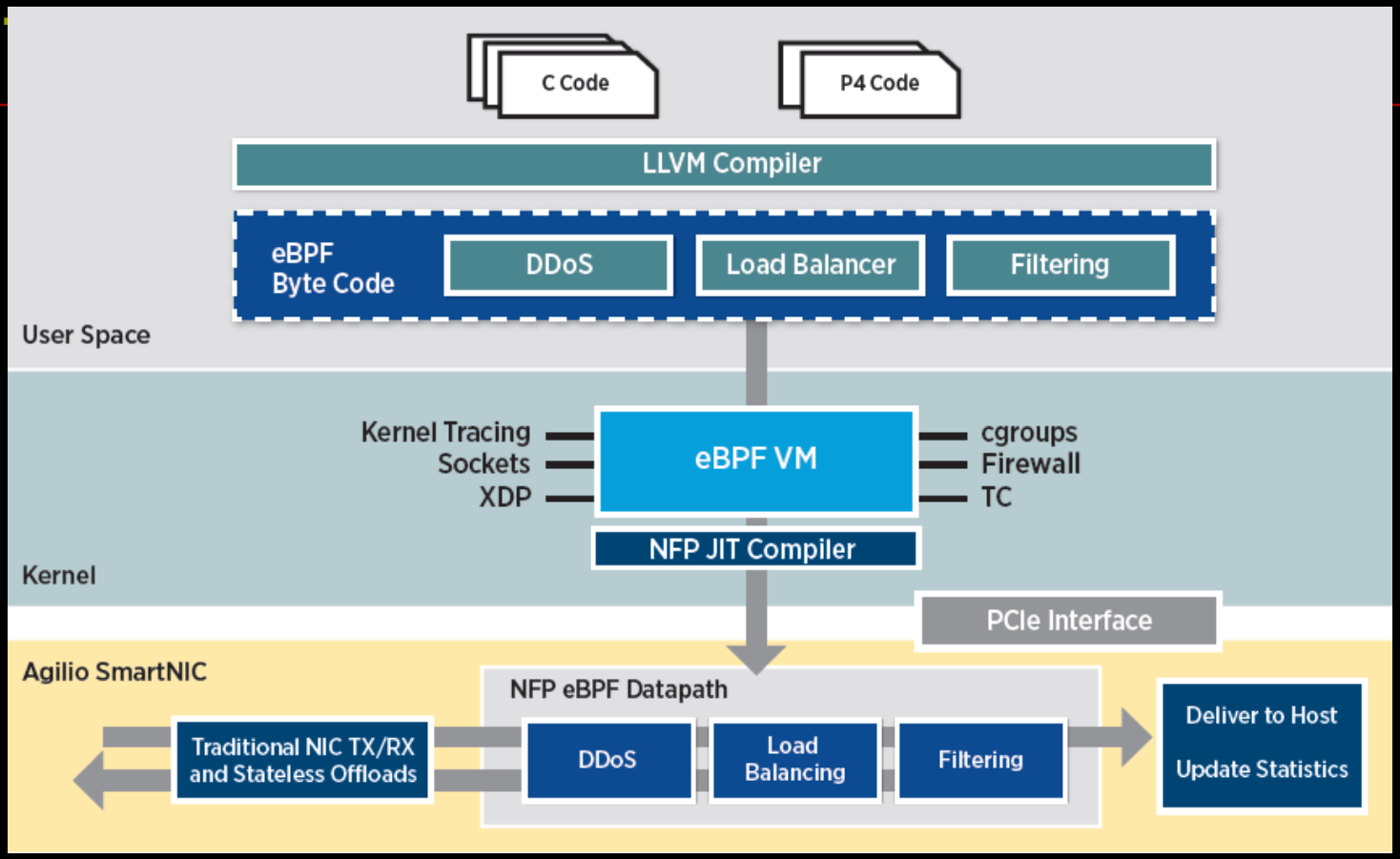
Overview



Source: https://hackmd.io/@Shawn5141/2022q1-final-project

2.1.3 eBPF/XDP Hardware Offload to SmartNIC/DPU

2.1.3.1 Netronome



Source: https://www.netronome.com/media/documents/PB_Agilio-eBPF-7-20.pdf

- https://www.netronome.com/products/agilio-software/agilio-ebpf-software/
- https://qmonnet.github.io/whirl-offload/2021/09/23/bpftool-features-thread/
- ...

2.2 Security

LSM (Linux Security Module)

- https://en.m.wikipedia.org/wiki/Linux_Security_Modules
- https://www.kernel.org/doc/html/latest/admin-guide/LSM/index.html
- https://docs.kernel.org/security/lsm.html
- https://www.kernel.org/doc/html/latest/bpf/bpf_lsm.html
- https://docs.kernel.org/bpf/prog_lsm.html
- https://blog.cloudflare.com/live-patch-security-vulnerabilities-with-ebpf-lsm/
- …

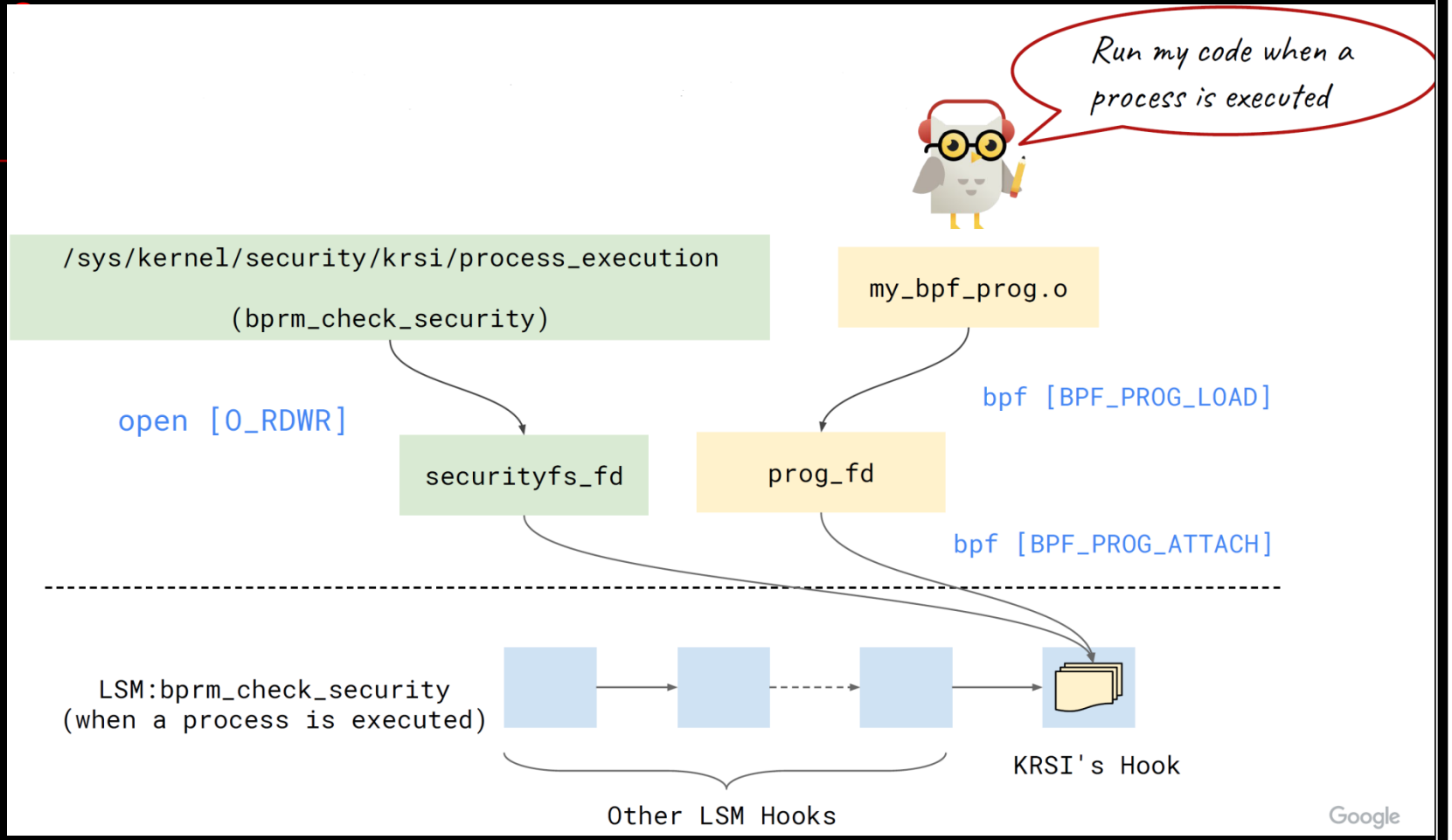
2.2.1 KRSI(Kernel Runtime Security Instrumentation)

Overview

- **eBPF + LSM**

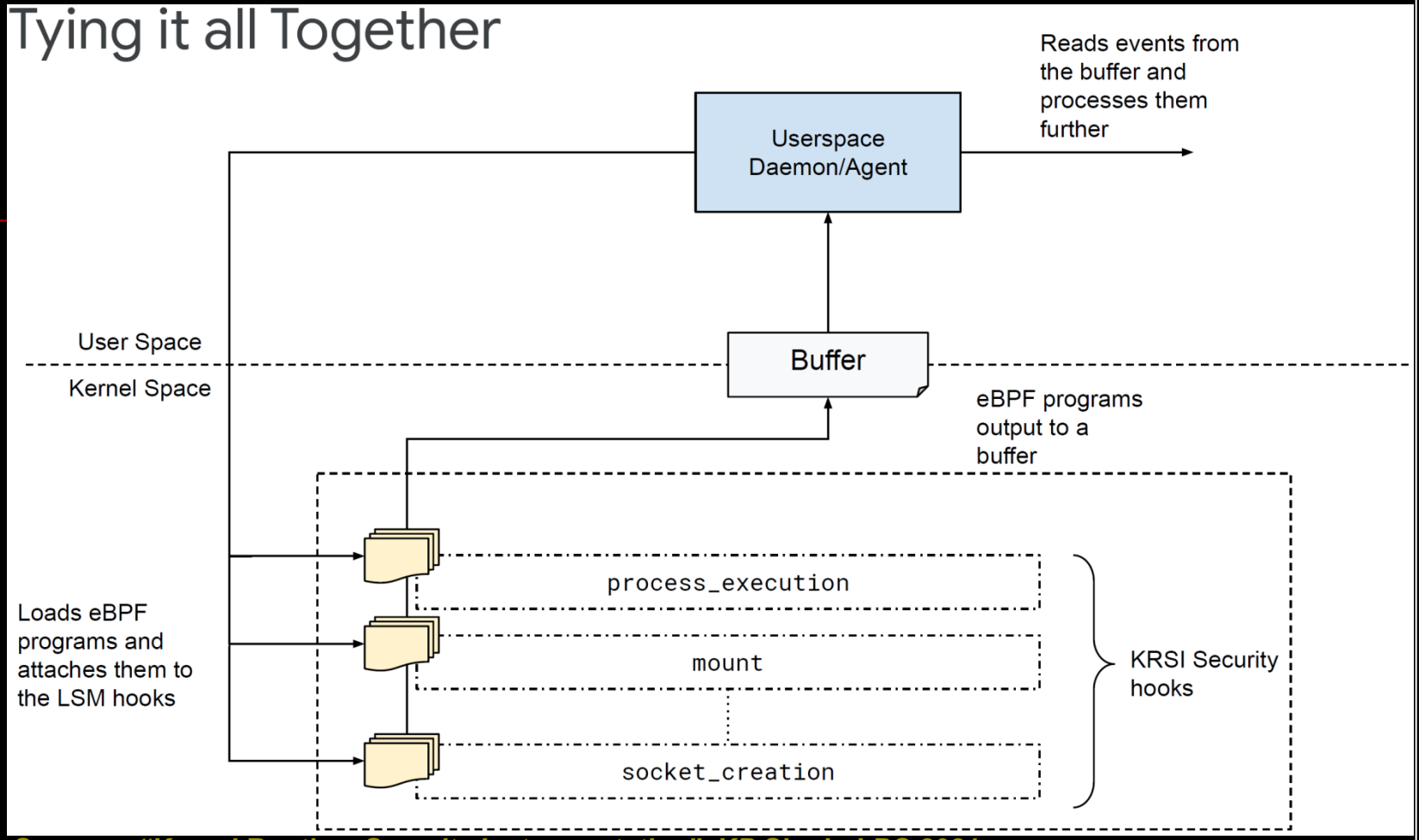
  https://lwn.net/Articles/798157/ //Kernel runtime security instrumentation
  https://lwn.net/Articles/808048/ //KRSI — the other BPF security module
  https://lwn.net/Articles/809841/ //KRSI and proprietary BPF programs
  https://lwn.net/Articles/813261/ //Impedance matching for BPF and LSM

How it works



Source: "Kernel Runtime Security Instrumentation", KP Singh, LPC 2021.

Tying it all Together



Source: "Kernel Runtime Security Instrumentation", KP Singh, LPC 2021.

2.3 Storage

2.3.1 eBPF-based in-kernel storage

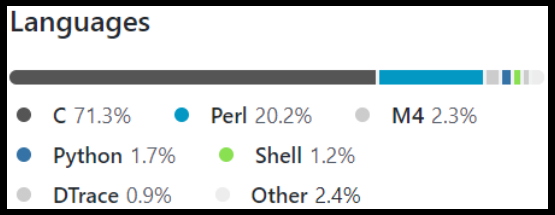
2.3.1.1 BMC

- https://github.com/Orange-OpenSource/bmc-cache

In-kernel cache based on eBPF.

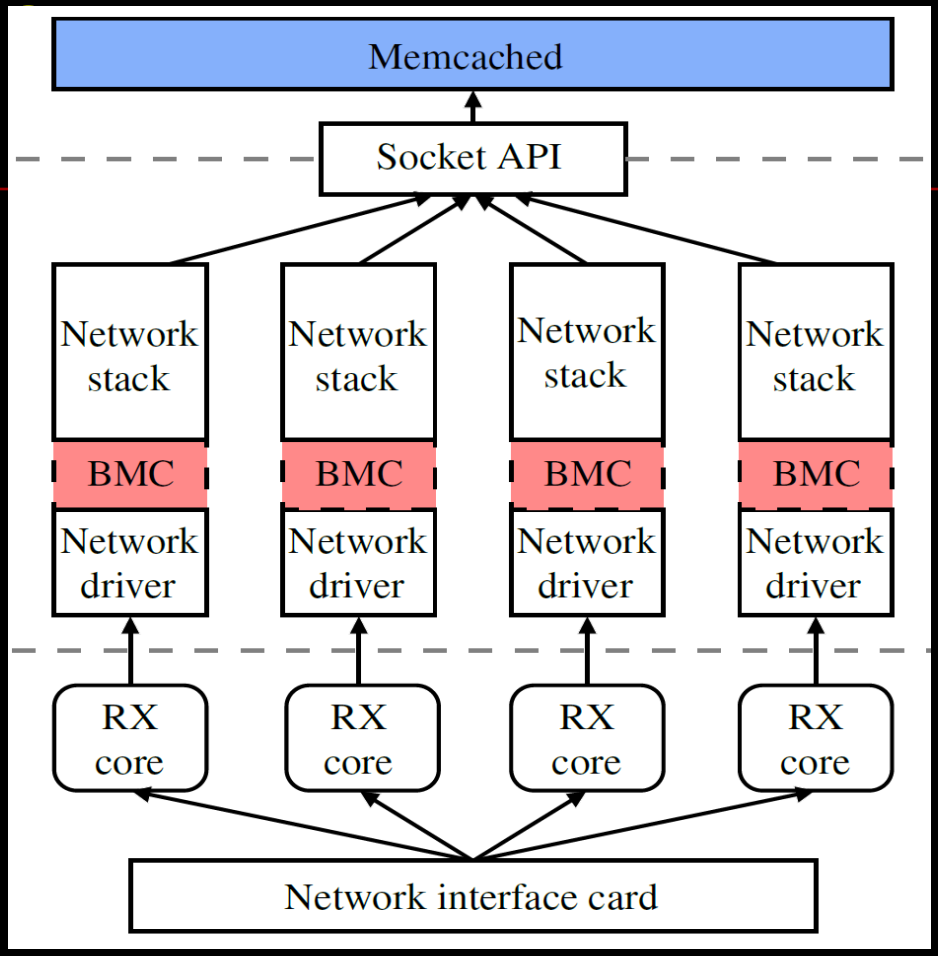
BMC (BPF Memory Cache) is an in-kernel cache for memcached. It enables runtime, crash-safe extension of the Linux kernel to process specific memcached requests before the execution of the standard network stack. BMC does not require modification of neither the Linux kernel nor the memcached application. Running memcached with BMC improves throughput by up to 18x compared to the vanilla memcached application.

- **Languages**



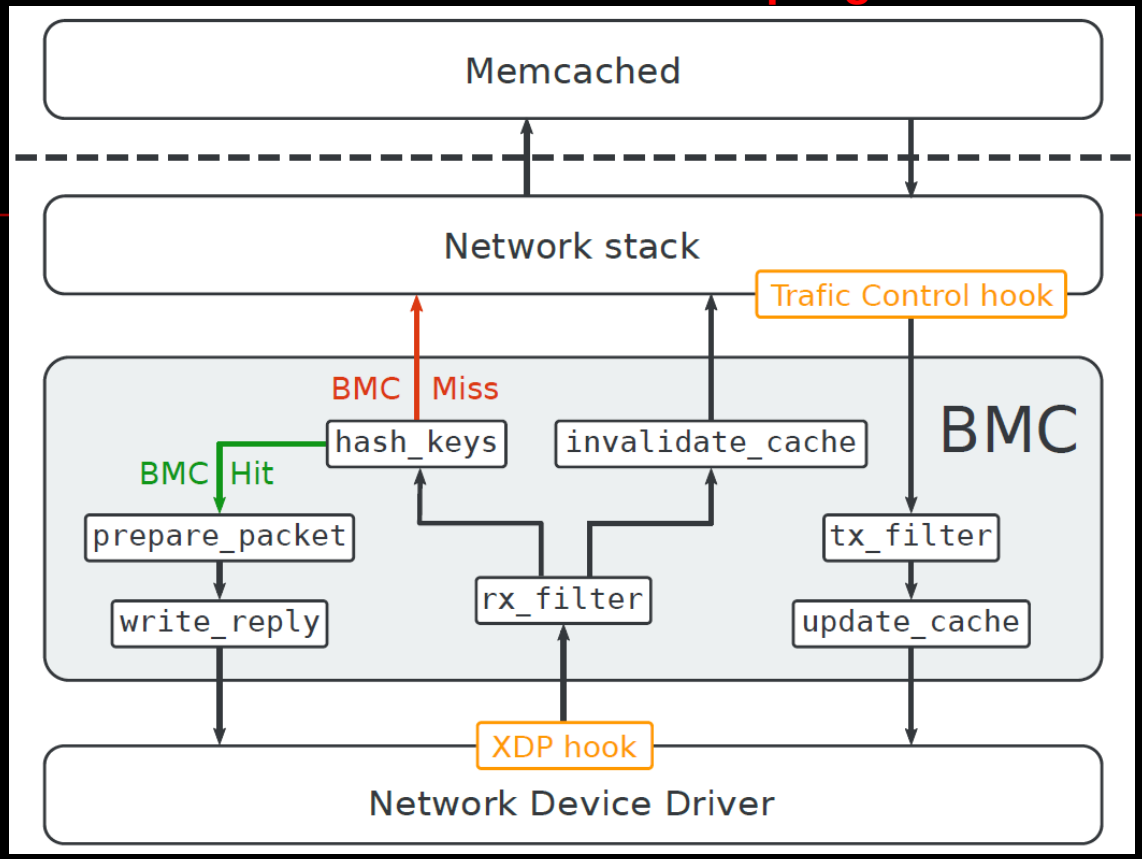
- …

Architecture & Design



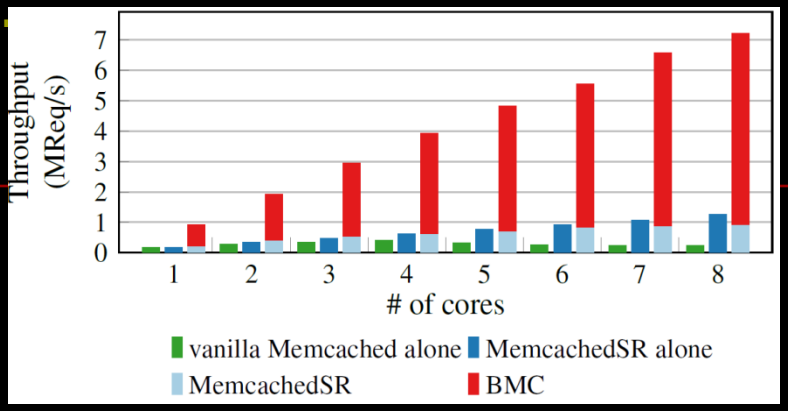
Source: https://www.usenix.org/system/files/nsdi21-ghigoff.pdf

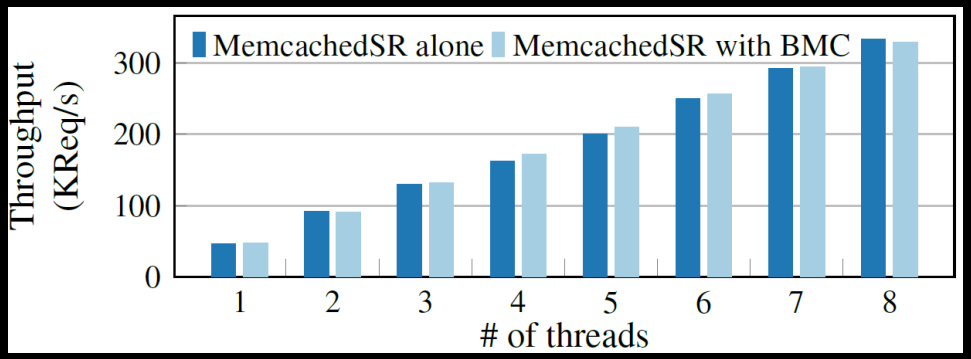
Division of BMC into seven eBPF programs

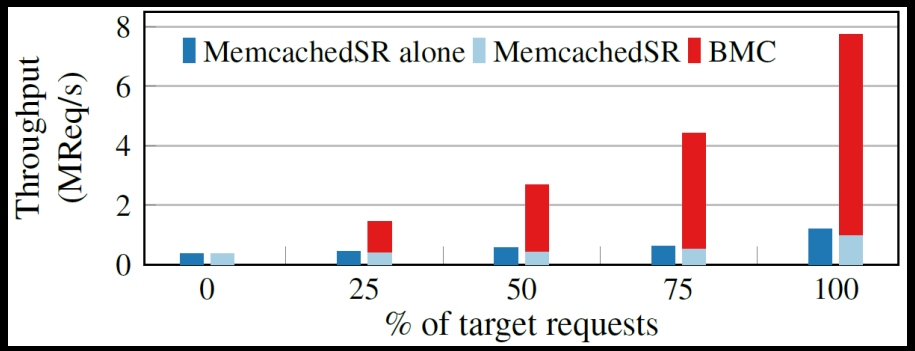


Source: https://www.usenix.org/system/files/nsdi21-ghigoff.pdf

Throughput







Source: https://www.usenix.org/system/files/nsdi21-ghigoff.pdf

2.3.2 eBPF-powered userland filesystems

2.3.2.1 FUSE BPF

- https://linuxplumbersconf.org/event/11/contributions/1048/
  Experimental, soon on LKML…
- FUSE BPF: stacking fs + passthrough + extFUSE ?

Implement a **generic stacking file system**

Allow **requests** to either be **handled by FUSE or** the **backing file system**

Allow **pre** and **post filtering** of backing file system request

Filtering can be either **by the kernel**, **or** through FUSE-style requests to **userspace**

Overcome **FUSE passthrough limitations** (per-file, read/write/mmap only)

Inspiration from

- **extFUSE**, presented by Ashish Bijlani at Plumbers in 2019 (https://linuxplumbersconf.org/event/4/contributions/415/)
- **FUSE passthrough**
- Stacking file systems, e.g., **incremental fs**

Source: "FS stacking with FUSE: performance issues and mitigations", Alessio Balsini & Paul Lawrence, LPC 2021.

- https://source.android.com/docs/core/storage/fuse-passthrough
- https://android.googlesource.com/platform/system/bpfprogs/

At a glance

Add to `fuse_inode`:

- `struct inode *backing_inode;`
- `struct bpf_prog *bpf;`

These may be set at mount time for root, at lookup time for all other inodes

If backing_inode exists, **all** requests will be conditionally sent **to the backing inode**, **else** we are in **classic FUSE** mode

If **no bpf**: simply forward as is (pure **passthrough** mode)

If **bpf**: format `fuse_args` with `in_args` and send to BPF, which may redirect request to **classic FUSE or**

1. Optionally request user-mode pre-filter with same modifiable in_args
2. (Potentially modified) request is sent to backing file system
3. Optionally pass `in_args` & `out_args` to BPF post-filter
4. Optionally pass `in_args` & `out_args` to user-mode post-filter

Early prototypes being tested within Android team

Source: "FS stacking with FUSE: performance issues and mitigations", Alessio Balsini & Paul Lawrence, LPC 2021.

Some thoughts

FUSE passthrough	FUSE BPF
How can we do better for Linux?	**BPF is a good compromise between user space and kernel space (good fit for FUSE)**
Do we really want *FUSE_PASSTHROUGH_CLOSE*?	**Would the Linux community benefit from this?**
Can be done with a mapping container (e.g., *IDR*), but is not as simple as *fuse2* (extra spinlocks)	**Is such architecture upstreamable?**

Source: "FS stacking with FUSE: performance issues and mitigations", Alessio Balsini & Paul Lawrence, LPC 2021.

2.4 New uses

- https://www.theregister.com/2022/09/14/linux_ebpf/
- …

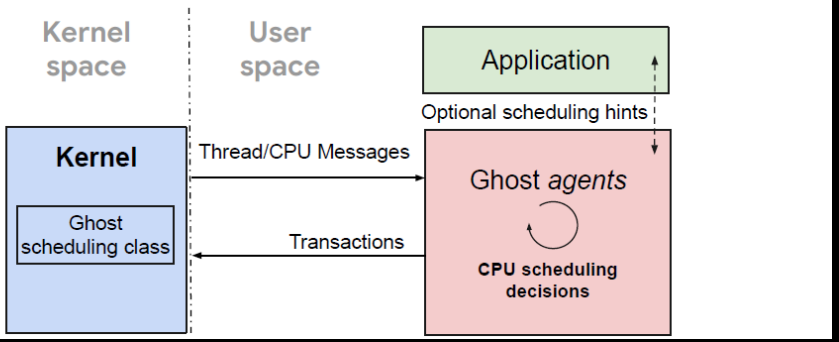
2.4.1 ghOSt

- **Fast & Flexible User-Space Delegation of Linux Scheduling**

ghOSt is a general-purpose delegation of scheduling policy implemented on top of the Linux kernel. The ghOSt framework provides a rich API that receives scheduling decisions for processes from userspace and actuates them as transactions. Programmers can use any language or tools to develop policies, which can be upgraded without a machine reboot. ghOSt supports policies for a range of scheduling objectives, from µs-scale latency, to throughput, to energy efficiency, and beyond, and incurs low overheads for scheduling actions. Many policies are just a few hundred lines of code. Overall, ghOSt provides a performant framework for delegation of thread scheduling policy to userspace processes that enables policy optimization, non-disruptive upgrades, and fault isolation.

- **https://github.com/google/ghost-userspace**
- **https://github.com/google/ghost-kernel**
- **What is it**

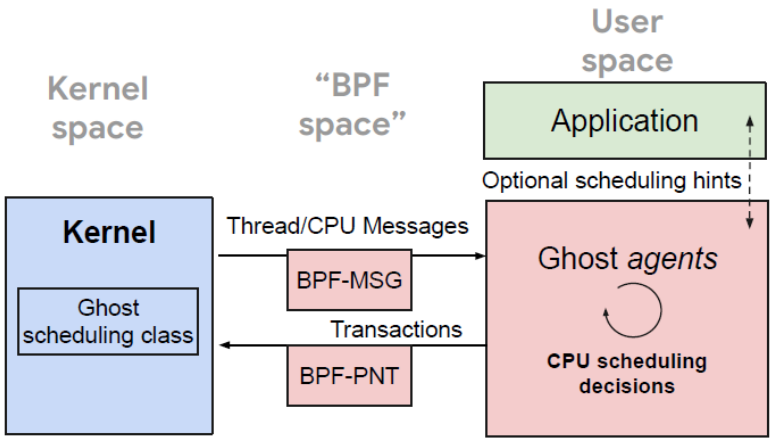
- Kernel scheduler class, below CFS in priority
- Scheduling decisions made in userspace by an *agent* process
- Kernel sends *messages* to the agent: “task X blocked on cpu 6”
- Agent issues *transactions* to the kernel: “run task X on cpu 12”



Source: https://lpc.events/event/16/contributions/1365/attachments/986/1912/lpc22-ebpf-kernel-scheduling-with-ghost.pdf

BPF in Ghost

- Agent process attaches a BPF program: BPF is an extension of the agent
- Messages -> BPF_GHOST_MSG_SEND
- Transactions -> BPF_GHOST_SCHED_PNT (pick_next_task)



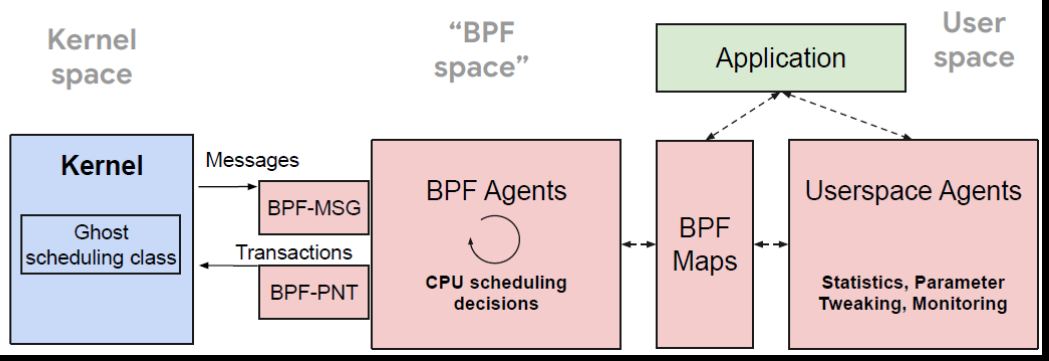
Ghost BPF Program Types: called from the kernel

- BPF-MSG: BPF_PROG_TYPE_GHOST_MSG
  - Context is *struct bpf_ghost_msg*
  - Attached at produce_for_task(struct task_struct *p, struct bpf_ghost_msg *msg)
  - e.g. MSG_TASK_WAKEUP: "task 6 woke on cpu 15"
- BPF-PNT: BPF_PROG_TYPE_GHOST_SCHED
  - Context is *struct bpf_ghost_sched*
  - Attached in pick_next_task_ghost()
  - Essentially picks the next task to run on this cpu, via a helper

Source: https://lpc.events/event/16/contributions/1365/attachments/986/1912/lpc22-ebpf-kernel-scheduling-with-ghost.pdf

"BPF-only" Scheduling

- All scheduling decisions are made in BPF
- Userspace has a role, but it is not in the critical path



Why Schedule in BPF instead of Userspace?

- Alternative: context switch to that cpu's agent task and let it handle messages and pick_next_task.
- Three reasons BPF is better:
  - No context switches! (Depends on your app if this matters)
  - Don't have to preempt a running task to run that cpu's agent.
    - e.g. Task 6 wakes up. Don't have to preempt another task to tell the agent about it.
  - BPF is synchronous! Solves a lot of heartache.
    - Hold the rq lock during bpf-msg, but not in bpf-pnt
    - In schedule()->pick_next_task() for bpf-pnt
- Downsides
  - Harder programming environment: limited loops, etc.
  - Event driven: harder to "spawn a background thread"
  - Data structures are limited to BPF Map types

Source: https://lpc.events/event/16/contributions/1365/attachments/986/1912/lpc22-ebpf-kernel-scheduling-with-ghost.pdf

2.4.2 HID

2.4.2.1 Input Devices

- https://www.phoronix.com/scan.php?page=news_item&px=Linux-eBPF-For-HID

Red Hat Eyeing Innovative eBPF Uses For Linux's HID Subsystem

Written by Michael Larabel in Linux Kernel on 24 February 2022 at 03:14 PM EST. 7 Comments

eBPF for sandboxed programs running in the kernel have shown to be very useful beyond the original BPF origins in the networking subsystem to also be very practical for other security, tracing, and other general use-cases for an in-kernel JIT virtual machine. Red Hat has sent out initial patches extending eBPF for making use of it within the HID subsystem for input devices.

Benjamin Tissoires of Red Hat's elite input Linux team has sent out a set of patches introducing eBPF support for HID devices. One of the very useful areas this eBPF support can be used within the HID area is for buggy/quirky devices. Currently there is lots of simple drivers and quirks for just correcting a key or byte in the report descriptor for input events. Unfortunately with the current approach with the input drivers being in the mainline kernel and the time it takes for upstreaming and getting down to vendor kernels is painful for users. The idea is that these "fixups" could be externalized in some external repository and ship these fixes as various eBPF programs that would be loaded at boot time to avoid needing a new kernel for quirky/buggy hardware.

- https://lpc.events/event/16/contributions/1364/attachments/959/1877/HID_BPF_slides_LPC22.pdf
- https://kernel-recipes.org/en/2022/talks/hid-bpf/

New Attempt

https://www.phoronix.com/scan.php?page=news_item&px=HID-eBPF-New-Attempt

Benjamin Tissoires of Red Hat has been the one spearheading the work on allowing eBPF usage within the HID subsystem. Making use of eBPF within HID could be very useful in dealing with quirky devices - currently there is lots of simple drivers and quirks for just correcting a key or byte in the report descriptor for input events. Unfortunately with the current approach with the input drivers being in the mainline kernel and the time it takes for upstreaming and getting down to vendor kernels is painful for users. The idea is that with eBPF'ing HID these "fix-ups" could be externalized in some external repository and ship these fixes as various eBPF programs that would be loaded at boot time to avoid needing a new kernel for quirky/buggy hardware. Usage of eBPF with HID could also allow for better supporting some newer/innovative devices like the Microsoft Surface Dial or USI stylus where the exposed kernel API is currently limited.

https:///lwn.net/Articles/886860/ //Introduce eBPF support for HID devices v1

…

https:///lwn.net/Articles/900909/ //Introduce eBPF support for HID devices v6

- **https://gitlab.freedesktop.org/bentiss/udev-hid-bpf**

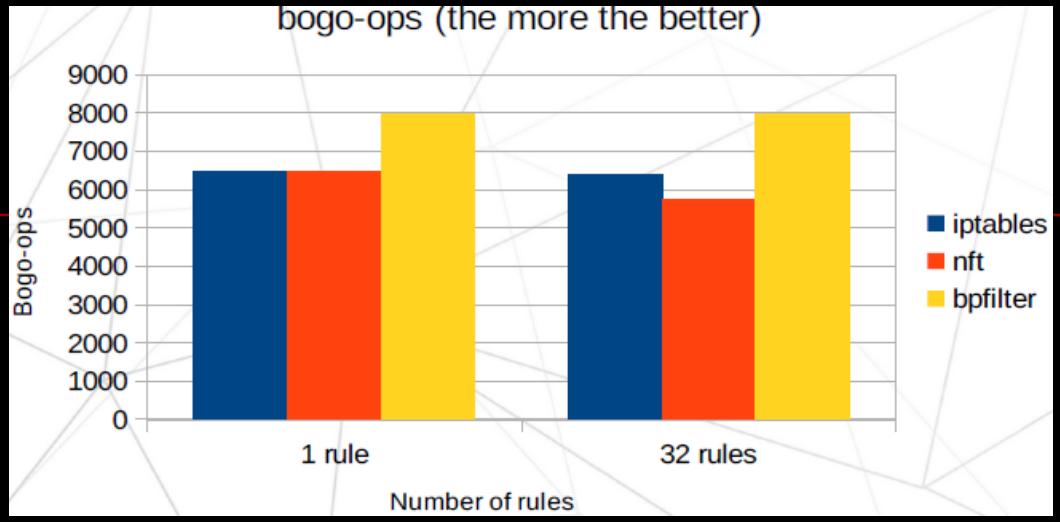
2.5 Comprehensive

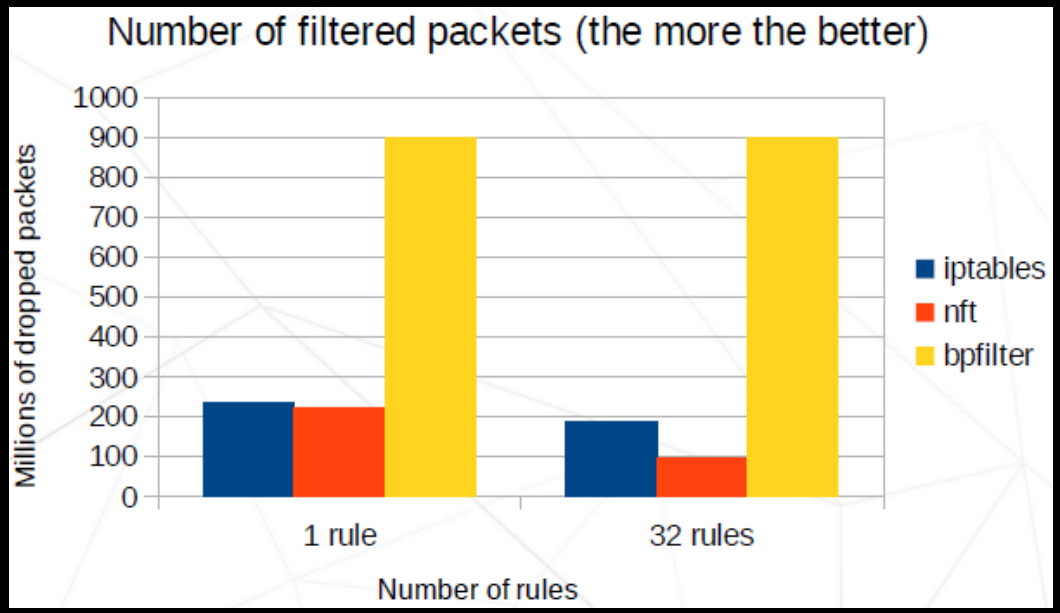
- ...

2.5.1 bpfilter

- https://www.mail-archive.com/netdev@vger.kernel.org/msg217095.html
- https://lwn.net/Articles/747504/ //net: add bpfilter
- https://lwn.net/Articles/749113/ //Re: [PATCH net-next] modules: allow modprobe load regular elf binaries
- https://lwn.net/Articles/749108/ //Designing ELF modules
- https://www.spinics.net/lists/netdev/msg486873.html //[RFC,POC 1/3] bpfilter: add experimental IMR bpf translator
- https://www.mail-archive.com/netdev@vger.kernel.org/msg217425.html //[PATCH RFC PoC 0/3] nftables meets bpf
- ...
- Replace iptables/nftables/netfiler with eBPF/bpfilter in the future
  https://lwn.net/Articles/755919/ //bpfilter (and user-mode blobs) for 4.18
  https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git/log/?qt=grep&q=bpfilter

Performance evaluation





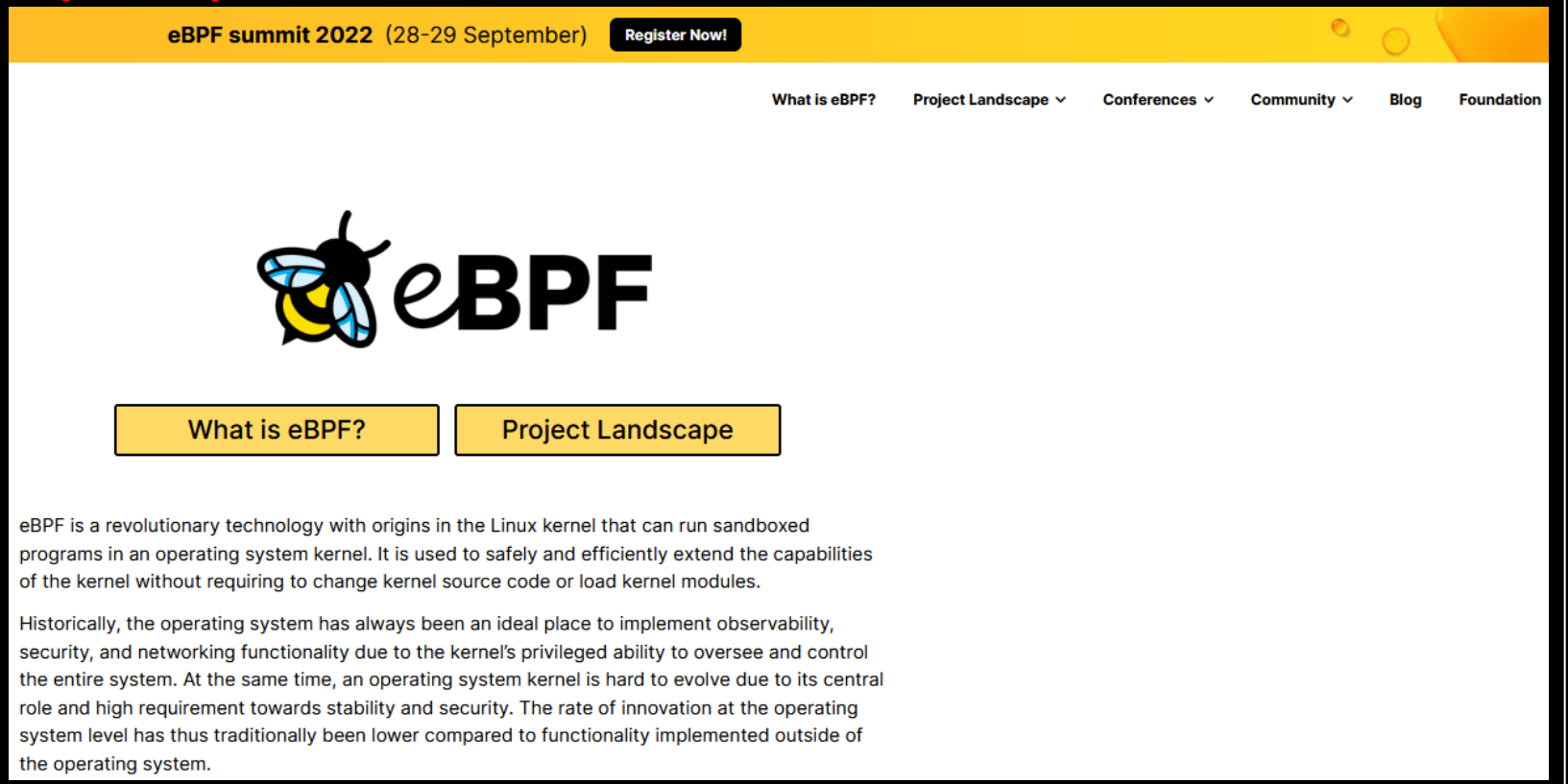
Source: "bpfilter – BPF based firewall", Dmitrii Banshchikov, LPC 2021.

III. eBPF ecosystem

1) What's new

1.1 Official Site and Foundation

- https://ebpf.io/



…

- **eBPF Foundation**

  https://www.linuxfoundation.org/press-release/facebook-google-isovalent-microsoft-and-netflix-launch-ebpf-foundation-as-part-of-the-linux-foundation/

  https://ebpf.io/foundation/

What is eBPF Foundation?

The number of eBPF-based projects has exploded in recent years and many more have been announcing intent to start adopting the technology. eBPF is quickly becoming one of the most influential technologies in the infrastructure software world. As such, the demand is high to optimize collaboration between projects and ensure that the core of eBPF is well maintained and equipped with a clear roadmap and vision for the bright future ahead of eBPF. This is where the eBPF Foundation comes in, and establishes an eBPF steering committee to take care of the technical direction and vision of eBPF.

If you are interested to collaborate with the eBPF Foundation, please join **#ebpf-foundation** on ebpf.io/slack.

eBPF Steering Committee

The eBPF Steering Committee (BSC) is responsible for the technical direction and overall vision of eBPF, the collaboration among projects, making recommendations to the governing board, defining the minimal requirements of eBPF runtimes, overseeing community events, maintaining project lifecycle procedures, and communicating on behalf of the eBPF community.

...

Platinum

Meta | Google | ISOVALENT | Microsoft

TIGERA | HUAWEI | CROWDSTRIKE

Silver

NETFLIX | DaoCloud | 数悦铭金 DATOMS | FUTUREWEI

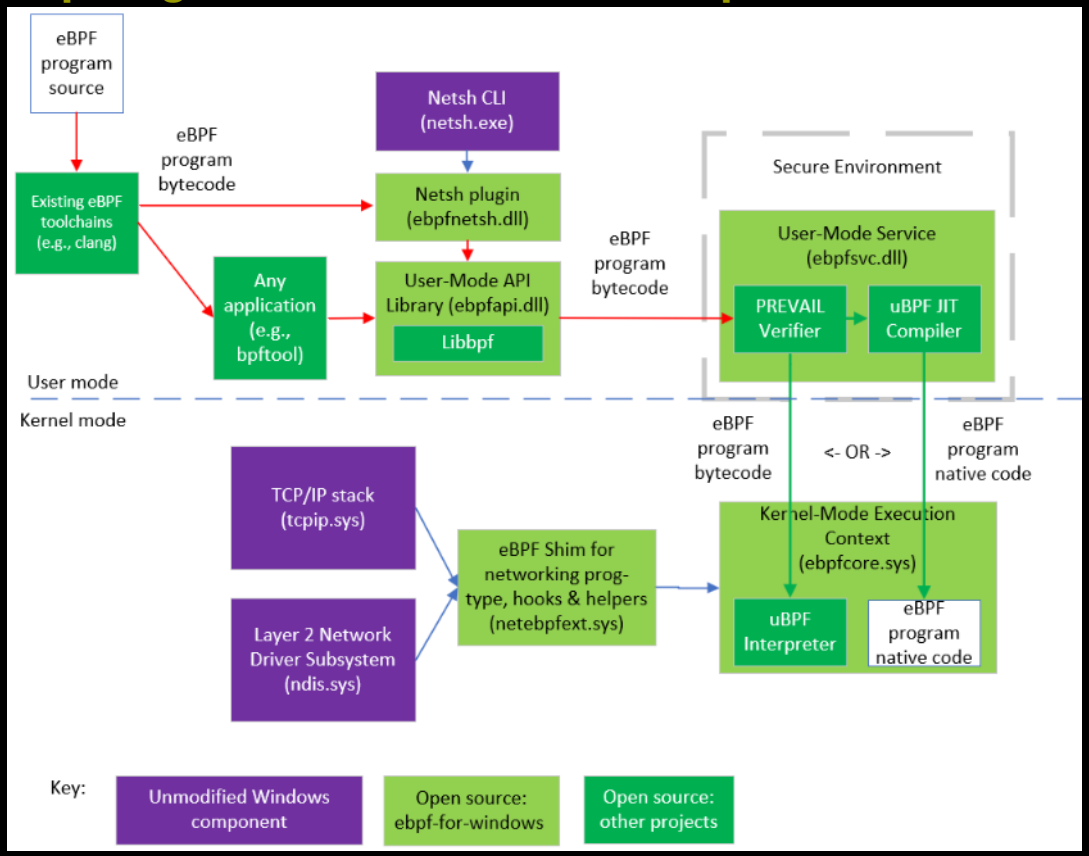
1.2 eBPF Conferences

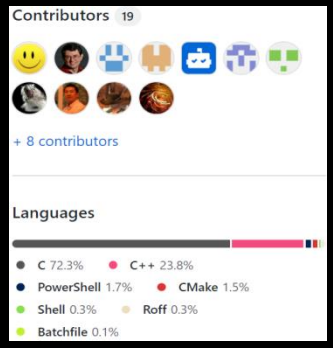
- eBPF Summit
  https://ebpf.io/summit-2022/
  https://ebpf.io/summit-2021/
  https://ebpf.io/summit-2020/
- Networking and BPF track at Linux Plumbers Conference
  https://lpc.events/event/16/sessions/131/#all //LPC 2022
  …
- Linux Kernel Developers' bpfconf
  http://vger.kernel.org/bpfconf2022.html
- Cloud Native eBPF Day
  https://events.linuxfoundation.org/cloud-native-ebpf-day-north-america/
  https://events.linuxfoundation.org/cloud-native-ebpf-day-europe/
  …
- More and more eBPF related topics are booming on various technical conferences.

1.3 Cross-Platform

eBPF on Windows

- https://cloudblogs.microsoft.com/opensource/2021/05/10/making-ebpf-work-on-windows/
- https://github.com/microsoft/ebpf-for-windows






```
[submodule "external/ebpf-verifier"]
        path = external/ebpf-verifier
        url = https://github.com/vbpf/ebpf-verifier.git
[submodule "external/ubpf"]
        path = external/ubpf
        url = https://github.com/iovisor/ubpf.git
        branch = master
[submodule "external/bpftool"]
        path = external/bpftool
        url = https://github.com/dthaler/bpftool-1.git
        branch = windows
[submodule "external/win-c"]
        path = external/win-c
        url = https://github.com/takamin/win-c.git
        branch = windows
[submodule "external/Catch2"]
        path = external/Catch2
        url = https://github.com/catchorg/Catch2.git
[submodule "external/FindWDK"]
        path = external/FindWDK
        url = https://github.com/SergiusTheBest/FindWDK
[submodule "external/pe-parse"]
        path = external/pe-parse
        url = https://github.com/trailofbits/pe-parse.git
```

As shown in the diagram, existing eBPF toolchains (clang, etc.) can be used to generate eBPF bytecode from source code in various languages. Bytecode can be consumed by any application, or via bpftool or the Netsh command line tool, which use a shared library that exposes Libbpf APIs, though this is still in progress.

The eBPF bytecode is sent to a static verifier (the PREVAIL verifier) that is hosted in a secure user-mode environment such as a system service (which is the case at present), enclave, or trusted VM. If the eBPF program passes all the verifier checks, it can be loaded into the kernel-mode execution context. Typically this is done by being JIT compiled (via the uBPF JIT compiler) into native code that is passed to the execution context. In a debug build, the byte code can instead be directly loaded into an interpreter (from uBPF in the kernel-mode execution context) though the interpreter is not present in a release build as it is considered less secure. See also the HVCI FAQ answer below.

eBPF programs installed into the kernel-mode execution context can attach to various hooks and call various helper APIs exposed by the eBPF shim, which internally wraps public Windows kernel APIs, allowing the use of eBPF on existing versions of Windows. Many helpers already exist, and more hooks and helpers will be added over time.

FAQ:

1. Is this a fork of eBPF?

No.

The eBPF for Windows project leverages existing projects, including the IOVisor uBPF project and the PREVAIL verifier, running them on top of Windows by adding the Windows-specific hosting environment for that code.

2. Does this provide app compatibility with eBPF programs written for Linux?

The intent is to provide source code compatibility for code that uses common hooks and helpers that apply across OS ecosystems.

Linux provides many hooks and helpers, some of which are very Linux specific (e.g., using Linux internal data structs) that would not be applicable to other platforms. Other hooks and helpers are generically applicable and the intent is to support them for eBPF programs.

Similarly, the eBPF for Windows project exposes Libbpf APIs to provide source code compatibility for applications that interact with eBPF programs.

3. Will eBPF work with HyperVisor-enforced Code Integrity (HVCI)?

Yes. With HVCI enabled, eBPF programs cannot be JIT compiled, but can be run either natively or in interpreted mode (but the interpreter is disabled in release builds and is only supported in debug builds). To understand why JIT compiled mode does not work, we must first understand what HVCI does.

HyperVisor-enforced Code Integrity (HVCI) is a mechanism whereby a hypervisor, such as Hyper-V, uses hardware virtualization to protect kernel-mode processes against the injection and execution of malicious or unverified code. Code integrity validation is performed in a secure environment that is resistant to attack from malicious software, and page permissions for kernel mode are set and maintained by the hypervisor.

Since a hypervisor doing such code integrity checks will refuse to accept code pages that aren't signed by a key that the hypervisor trusts, this does impact eBPF programs running natively. As such, when HVCI is enabled, eBPF programs work fine in interpreted mode, but not when using JIT compilation because the JIT compiler does not have a key that the hypervisor trusts. And since interpreted mode is absent in release builds, neither mode will work on an HVCI-enabled production system.

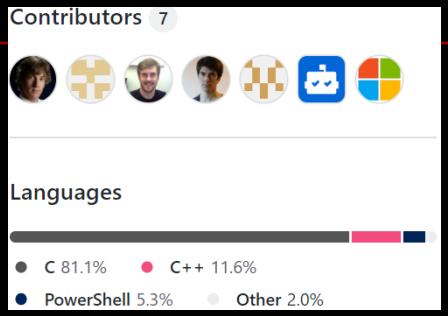
Instead, a third mode is also supported by eBPF for Windows, in addition to JIT compiled and interpreted modes. This third mode entails compiling eBPF programs into regular Windows drivers that can be accepted by HVCI. For more discussion, see the Native Code Generation documentation.

- **https://lwn.net/Articles/857215/ //Implementing eBPF for Windows**
- **https://thenewstack.io/microsoft-brings-ebpf-to-windows/**
- **…**

XDP For Windows

- https://github.com/microsoft/xdp-for-windows/

"XDP for Windows" is a Windows interface similar to XDP (eXpress Data Path), used to send and receive packets at high rates by bypassing most of the OS networking stack.




```
1   [submodule "submodules/ndis-driver-library"]
2           path = submodules/ndis-driver-library
3           url = https://github.com/microsoft/ndis-driver-library
4   [submodule "submodules/wil"]
5           path = submodules/wil
6           url = https://github.com/microsoft/wil
```

- https://github.com/microsoft/xdp-for-windows/blob/main/docs/faq.md

> Is this a fork of the Linux XDP project?

No. We've been heavily inspired and influenced by the Linux XDP project, but we don't reuse any Linux code.

> Do you support AF_XDP?

Yes, we provide an AF_XDP API sockets API. This API does not rely on Winsock; it is an independent socket API. It is not directly source-compatible with the Linux API, though it is possible to build a thin shim layer to abstract any differences.

> Do you support eBPF programs?

Not yet. We plan to integrate with the eBPF for Windows project soon. In the meantime, we have built a barebones program module.

> What versions of Windows does XDP support?

XDP is currently tested on Windows Server 2019 and 2022.

> When is this shipping with Windows?

At this time, there is no plan to ship XDP as a part of Windows. XDP will release separately from Windows, likely directly through GitHub.

- https://cloud7.news/development/microsoft-introduced-open-source-xdp-for-windows/
- …

Status

Platform →	Linux		MacOSX		Android		FreeBSD		Windows		TockOS (embedded)	
↓ Project	Kernel	User	Kernel	User	Kernel	User	Kernel	User	Kernel	User	Kernel	User
Linux	2014											
uBPF		2015										
rbpf		2017		2017						2018		
Android					2017							
Generic eBPF	2017	2017		2017			2017	2017				
eBPF for Windows									2021			
Tock											2021	

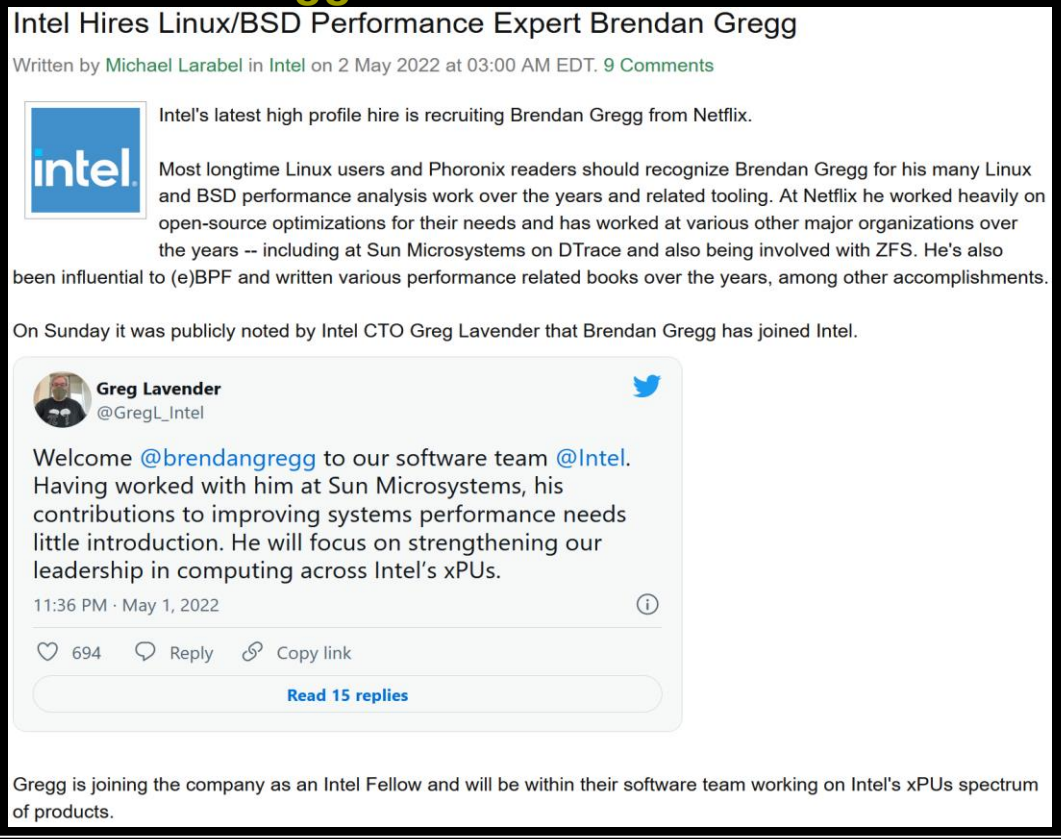
Source: "The Cross-Platform Future of eBPF", Dave Thaler(Microsoft), Cloud Native eBPF Day North America 2021

1.4 Hall Of Fame

- …

Brendan Gregg

- https://www.brendangregg.com/
- https://www.phoronix.com/scan.php?page=news_item&px=Brendan-Gregg-Intel



Intel Hires Linux/BSD Performance Expert Brendan Gregg

Written by Michael Larabel in Intel on 2 May 2022 at 03:00 AM EDT. 9 Comments

Intel's latest high profile hire is recruiting Brendan Gregg from Netflix.

Most longtime Linux users and Phoronix readers should recognize Brendan Gregg for his many Linux and BSD performance analysis work over the years and related tooling. At Netflix he worked heavily on open-source optimizations for their needs and has worked at various other major organizations over the years -- including at Sun Microsystems on DTrace and also being involved with ZFS. He's also been influential to (e)BPF and written various performance related books over the years, among other accomplishments.

On Sunday it was publicly noted by Intel CTO Greg Lavender that Brendan Gregg has joined Intel.

Greg Lavender
@GregL_Intel

Welcome @brendangregg to our software team @Intel. Having worked with him at Sun Microsystems, his contributions to improving systems performance needs little introduction. He will focus on strengthening our leadership in computing across Intel's xPUs.

11:36 PM · May 1, 2022

694 Reply Copy link

Read 15 replies

Gregg is joining the company as an Intel Fellow and will be within their software team working on Intel's xPUs spectrum of products.

2) Development
2.1 Toolchain
2.1.1 LLVM

- eBPF backend firstly introduced in LLVM 3.7 release
- http://llvm.org/docs/CodeGenerator.html#the-extended-berkeley-packet-filter-ebpf-backend
-
  - Enabled by default with all major distributions
    - Registered targets: `llc --version`
    - `llc`'s BPF `-march` options: bpf, bpfeb, bpfel
    - `llc`'s BPF `-mcpu` options: generic, v1, v2, probe
  - Assembler output through `-S` supported
  - `llvm-objdump` for disassembler and code annotations (via DWARF)
  - Annotations correlate directly with kernel verifier log
  - Outputs ELF file with maps as relocation entries
    - Processed by BPF loaders (e.g. iproute2) and pushed into kernel

Source: https://ossna2017.sched.com/event/BCsg/making-the-kernels-networking-data-path-programmable-with-bpf-and-xdp-daniel-borkmann-covalent

- $LLVM_SRC/lib/Target/BPF
- http://cilium.readthedocs.io/en/latest/bpf/
- LLVM 9.0 released with ability to build the Linux x86_64

2.1.2 GCC

- https://lwn.net/Articles/831402/ //BPF in GCC
- …

2.2 libbpf and bpftool

2.2.1 libbpf

- https://github.com/libbpf/libbpf

Libbpf documentation can be found here. It's an ongoing effort and has ways to go, but please take a look and consider contributing as well.

Please check out libbpf-bootstrap and the companion blog post for the examples of building BPF applications with libbpf. libbpf-tools are also a good source of the real-world libbpf-based tracing tools.

See also "BPF CO-RE reference guide" for the coverage of practical aspects of building BPF CO-RE applications and "BPF CO-RE" for general introduction into BPF portability issues and BPF CO-RE origins.

All general BPF questions, including kernel functionality, libbpf APIs and their application, should be sent to bpf@vger.kernel.org mailing list. You can subscribe to it here and search its archive here. Please search the archive before asking new questions. It very well might be that this was already addressed or answered before.

bpf@vger.kernel.org is monitored by many more people and they will happily try to help you with whatever issue you have. This repository's PRs and issues should be opened only for dealing with issues pertaining to specific way this libbpf mirror repo is set up and organized.

- Languages

C 97.5% Shell 2.1% Other 0.4%

- https://libbpf.readthedocs.io/en/latest/api.html
- https://docs.kernel.org/bpf/libbpf/index.html

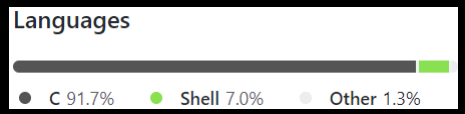
2.2.2 bpftool

- $KERNEL_SRC/tools/bpf/bpftool
- $KERNEL_SRC/tools/bpf/bpftool/Documentation
- https://github.com/libbpf/bpftool

This is a mirror of bpf-next Linux source tree's `tools/bpf/bpftool` directory, plus its few dependencies from under `kernel/bpf/`, and its supporting header files.

All the gory details of syncing can be found in `scripts/sync-kernel.sh` script.

Some header files in this repo (`include/linux/*.h`) are reduced versions of their counterpart files at bpf-next's `tools/include/linux/*.h` to make compilation successful.



Dependencies

Required:

- libelf
- zlib

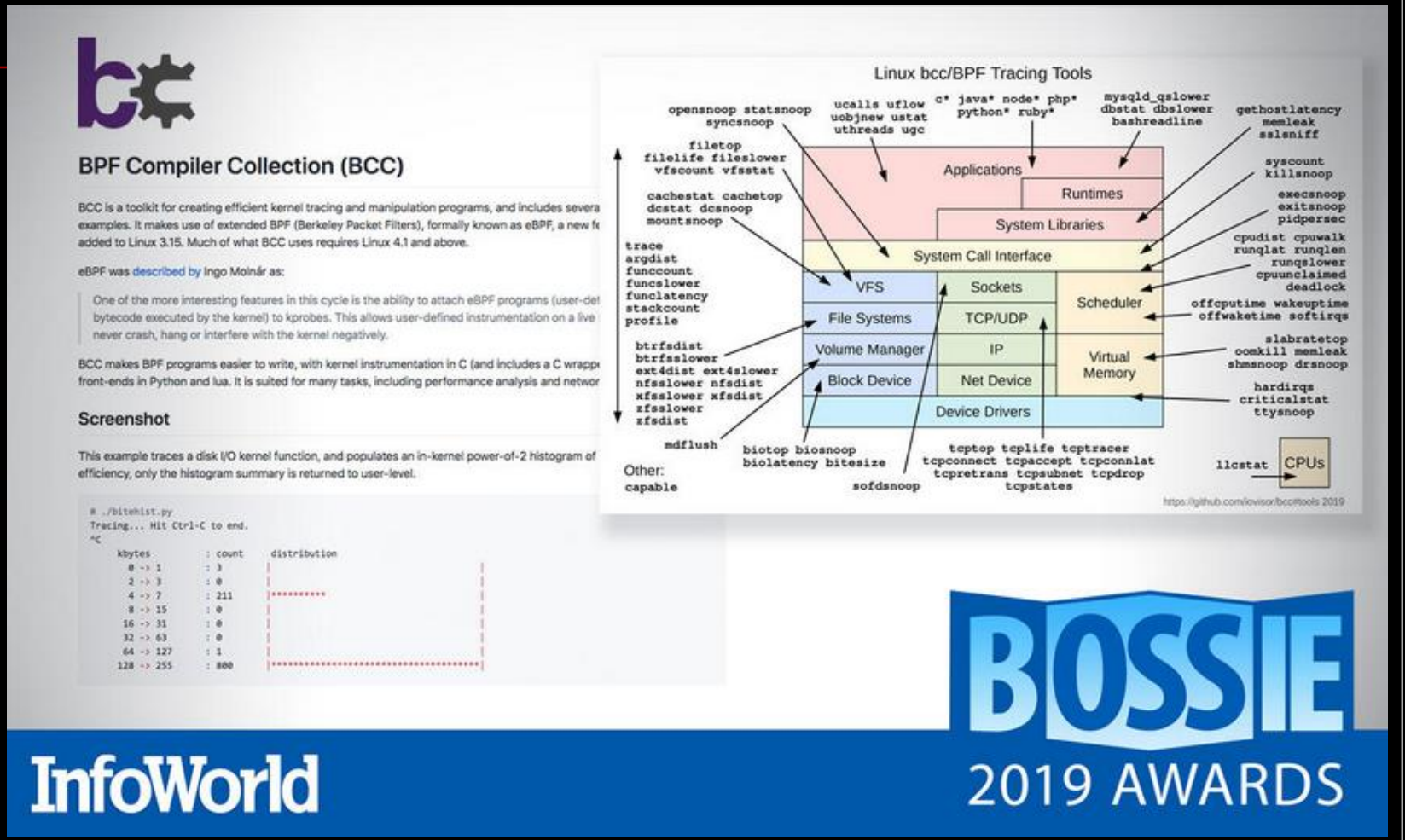
Optional:

- libbfd (for dumping JIT-compiled program instructions)
- libcap (for better feature probing)
- kernel BTF information (for profiling programs or showing PIDs of processes referencing BPF objects)
- clang/LLVM (idem)

- ...

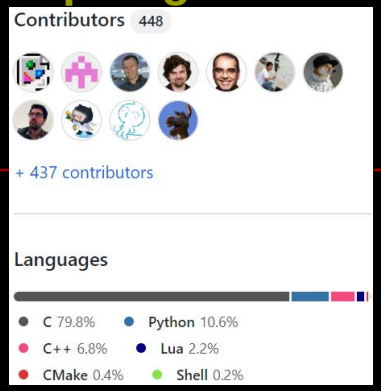
2.3 BCC (BPF Compiler Collection)

- https://www.infoworld.com/article/3444198/the-best-open-source-software-of-2019.html



What is it

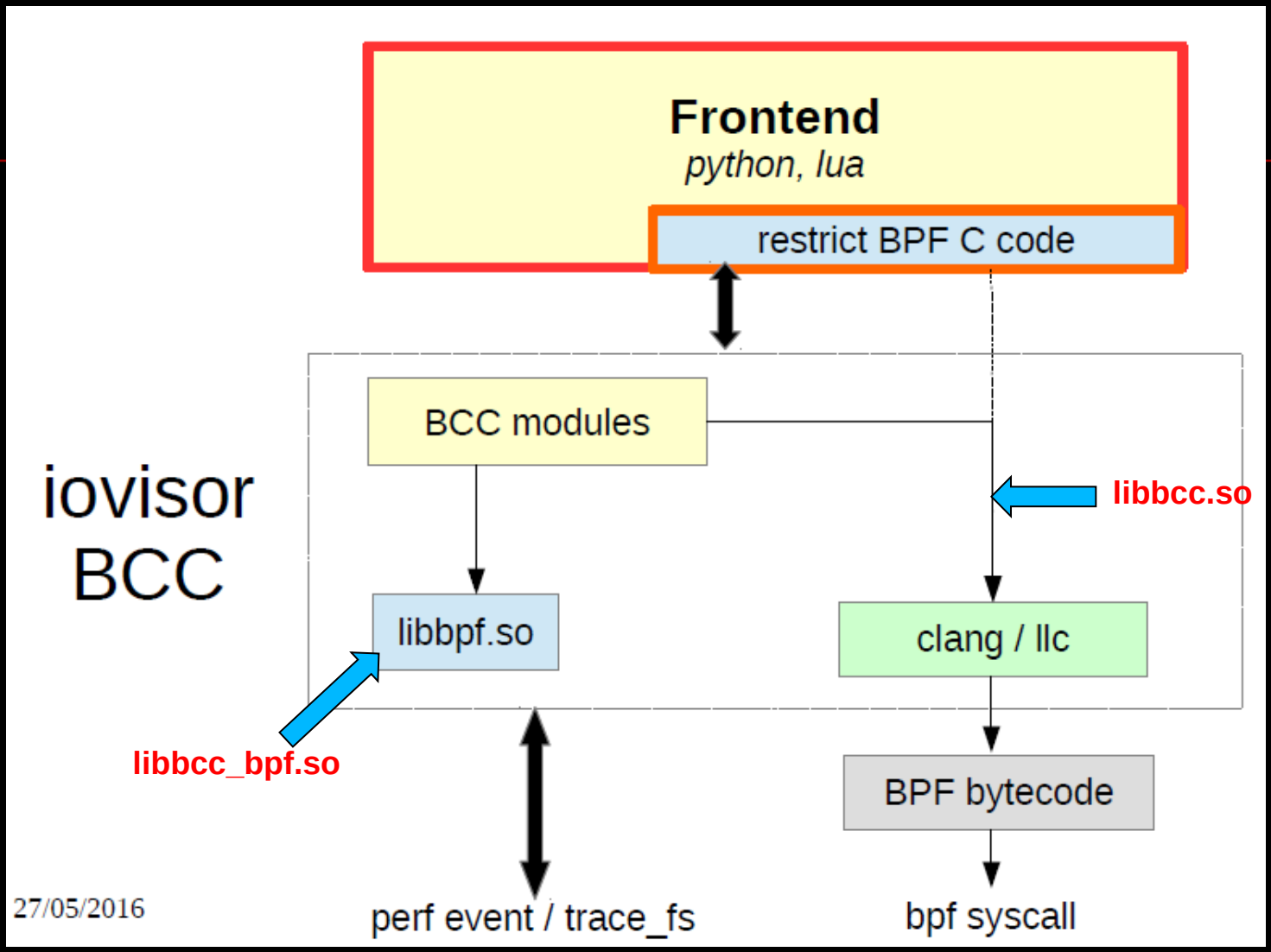
- https://github.com/iovisor/bcc/



A toolkit with Python/Lua frontend for compiling, loading, and executing BPF programs, which allows user-defined instrumentation on a live kernel image.

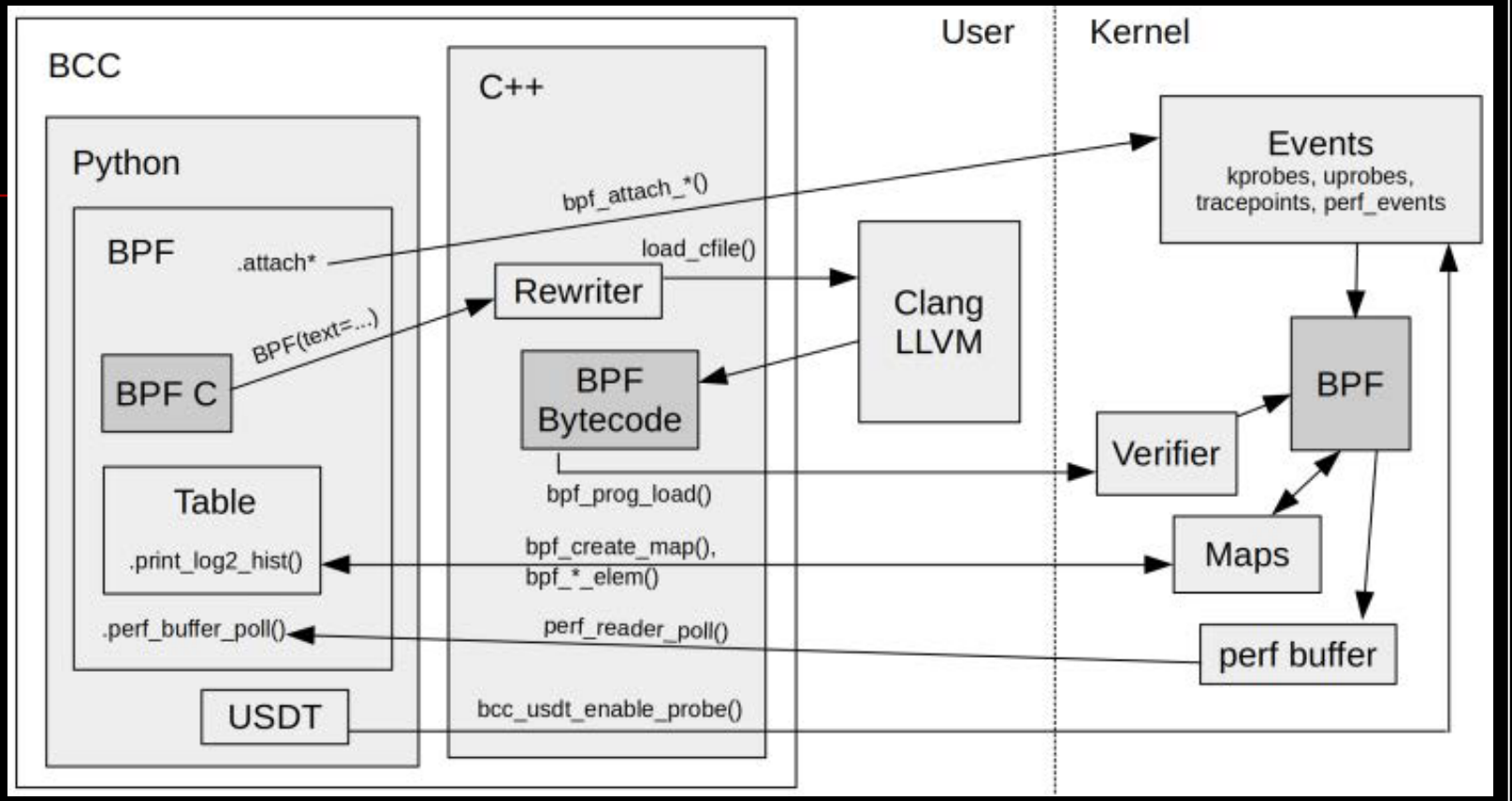
- Compile BPF program from C source(A modified C language for BPF backends).
- Attach BPF program to kprobe/uprobe/tracepoint/USDT/socket…
- Poll data from BPF program.
- Framework for building new tools or one-off scripts.
- Additional projects to support Go, Rust, and DTrace-style frontend.
- https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md
- …

Architecture & Design



Source: http://www.slideshare.net/vh21/meet-cutebetweenebpfandtracing

Internals

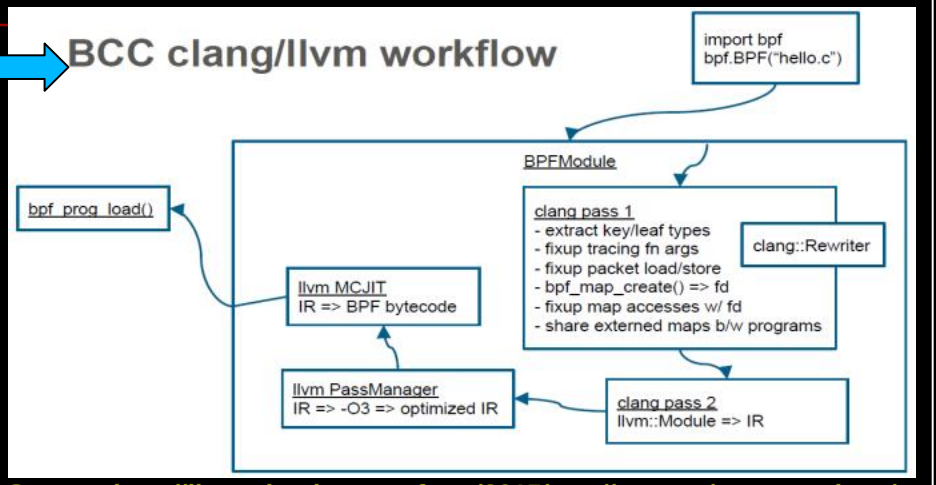


Source: https://brendangregg.com/bpf-performance-tools-book.html

How it works

- https://lwn.net/Articles/747640/ //Some advanced BCC topics

```
#!/usr/bin/env python

from bcc import BPF
from time import sleep

program = """
    BPF_HASH(callers, u64, unsigned long);

    TRACEPOINT_PROBE(kmem, kmalloc) {
        u64 ip = args->call_site;
        unsigned long *count;
        unsigned long c = 1;

        count = callers.lookup((u64 *)&ip);
        if (count != 0)
            c += *count;

        callers.update(&ip, &c);

        return 0;
    }
"""
b = BPF(text=program)

while True:
    try:
        sleep(1)
        for k,v in sorted(b["callers"].items()):
            print ("%s %u" % (b.ksym(k.value), v.value))
        print
    except KeyboardInterrupt:
        exit()
```



Source: http://linuxplumbersconf.org/2015/ocw//system/presentations/3249/original/bpf_llvm_2015aug19.pdf

The output from this little program looks like:

```
# ./example.py
i915_sw_fence_await_dma_fence 4
intel_crtc_duplicate_state 4
SyS_memfd_create 1
drm_atomic_state_init 4
sg_kmalloc 7
intel_atomic_state_alloc 4
seq_open 504
SyS_bpf 22
```

Python frontend

- https://github.com/iovisor/bcc/blob/master/examples/tracing/nodejs_http_server.py

```
#!/usr/bin/python
#
# nodejs_http_server    Basic example of node.js USDT tracing.
#                       For Linux, uses BCC, BPF. Embedded C.
#
# USAGE: nodejs_http_server PID
#
# Copyright 2016 Netflix, Inc.
# Licensed under the Apache License, Version 2.0 (the "License")

from __future__ import print_function
from bcc import BPF, USDT
from bcc.utils import printb
import sys

if len(sys.argv) < 2:
    print("USAGE: nodejs_http_server PID")
    exit()
pid = sys.argv[1]
debug = 0

# load BPF program
bpf_text = """
#include <uapi/linux/ptrace.h>
int do_trace(struct pt_regs *ctx) {
    uint64_t addr;
    char path[128]={0};
    bpf_usdt_readarg(6, ctx, &addr);
    bpf_probe_read_user(&path, sizeof(path), (void *)addr);
    bpf_trace_printk("path:%s\\n", path);
    return 0;
};
"""
```

```

# enable USDT probe from given PID
u = USDT(pid=int(pid))
u.enable_probe(probe="http__server__request", fn_name="do_trace")
if debug:
    print(u.get_text())
    print(bpf_text)

# initialize BPF
b = BPF(text=bpf_text, usdt_contexts=[u])

# header
print("%-18s %-16s %-6s %s" % ("TIME(s)", "COMM", "PID", "ARGS"))

# format output
while 1:
    try:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
    except ValueError:
        print("value error")
        continue
    except KeyboardInterrupt:
        exit()
    printb(b"%-18.9f %-16s %-6d %s" % (ts, task, pid, msg))
```

...

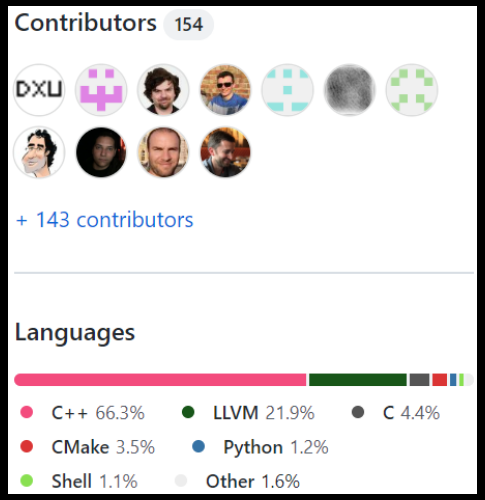
- https://github.com/iovisor/bcc/blob/master/docs/tutorial_bcc_python_developer.md

2.4 bpftrace

- https://github.com/iovisor/bpftrace

bpftrace is a high-level tracing language for Linux enhanced Berkeley Packet Filter (eBPF) available in recent Linux kernels (4.x). bpftrace uses LLVM as a backend to compile scripts to BPF-bytecode and makes use of BCC for interacting with the Linux BPF system, as well as existing Linux tracing capabilities: kernel dynamic tracing (kprobes), user-level dynamic tracing (uprobes), and tracepoints. The bpftrace language is inspired by awk and C, and predecessor tracers such as DTrace and SystemTap. bpftrace was created by Alastair Robertson.

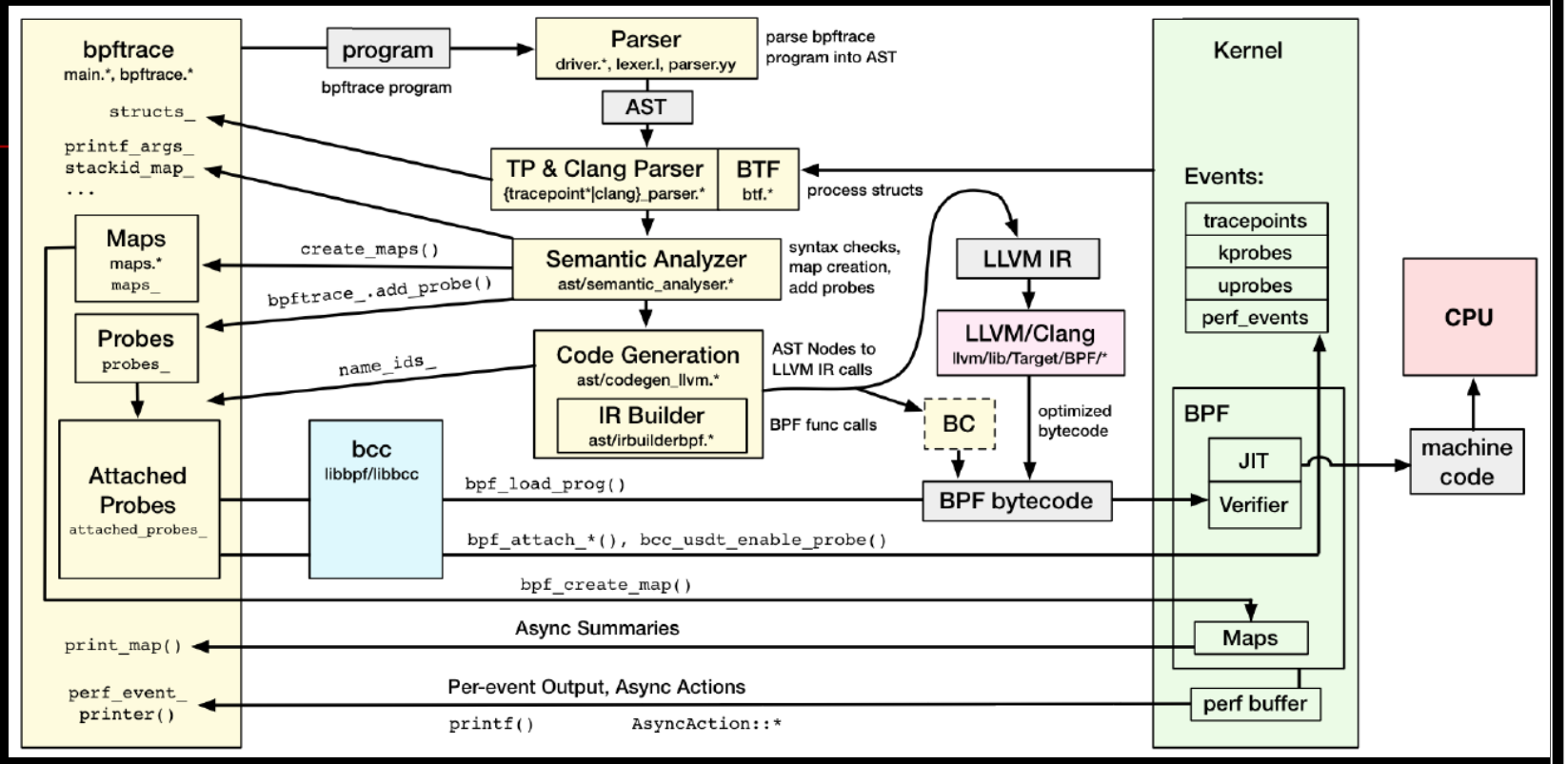
To learn more about bpftrace, see the Manual the Reference Guide and One-Liner Tutorial.



- Supported architectures
  x86_64, arm64 and s390x.

Architecture & Design

- Mid-level internals



Source: https://www.brendangregg.com/Slides/LISA2021_BPF_Internals

Examples

List all probes with "sleep" in their name

```
# bpftrace -l '*sleep*'
```

Trace processes calling sleep

```
# bpftrace -e 'kprobe:do_nanosleep { printf("%d sleeping\n", pid); }'
```

Trace processes calling sleep while spawning `sleep 5` as a child process

```
# bpftrace -e 'kprobe:do_nanosleep { printf("%d sleeping\n", pid); }' -c 'sleep 5'
```

Source: https://github.com/iovisor/bpftrace/blob/master/man/adoc/bpftrace.adoc#examples

- https://github.com/iovisor/bpftrace/tree/master/tools
- …

2.5 ply

- https://github.com/wkz/ply

Light-weight Dynamic Tracer for Linux.

A light-weight dynamic tracer for Linux that leverages the kernel's BPF VM in concert with kprobes and tracepoints to attach probes to arbitrary points in the kernel. Most tracers that generate BPF bytecode are based on the LLVM based BCC toolchain. ply on the other hand has no required external dependencies except for `libc`. In addition to `x86_64`, ply also runs on `aarch64`, `arm`, and `powerpc`. Adding support for more ISAs is easy.

`ply` follows the Little Language approach of yore, compiling ply scripts into Linux BPF programs that are attached to kprobes and tracepoints in the kernel. The scripts have a C-like syntax, heavily inspired by `dtrace(1)` and, by extension, `awk(1)`.

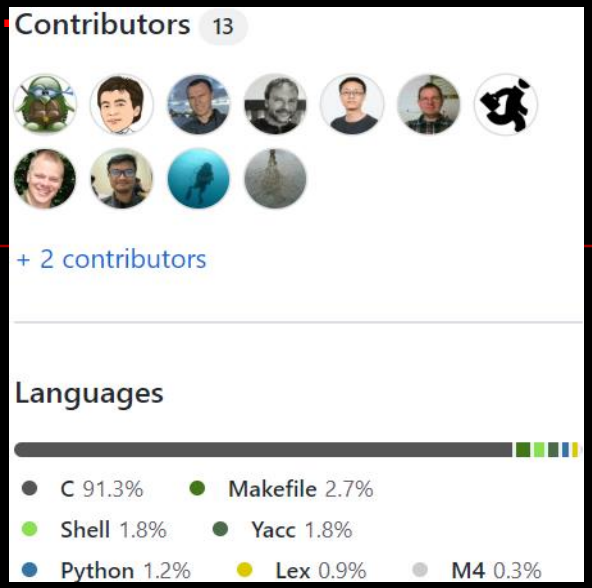
The primary goals of `ply` are:

- Expose most of the BPF tracing feature-set in such a way that new scripts can be whipped up very quickly to test different hypotheses.
- Keep dependencies to a minimum. Right now Flex and Bison are required at build-time, leaving `libc` as the only runtime dependency. Thus, `ply` is well suited for embedded targets.

If you need more fine-grained control over the kernel/userspace interaction in your tracing, checkout the bcc project which compiles C programs to BPF using LLVM in combination with a python userspace recipient to give you the full six degrees of freedom.

- The ply BPF front end, created by Tobias Waldekranz, provides a high-level language similar to bpftrace and requires minimal dependencies (no LLVM or Clang). This makes it suited to resource-constrained environments, with the drawback that struct navigation and including header files (as required by many tools in this book) are not possible.

Source: https://www.brendangregg.com/bpf-performance-tools-book.html



- https://wkz.github.io/ply/

Examples

What is the distribution of the returned sizes from `read(2)`s to the VFS?

```
ply 'kretprobe:vfs_read { @["size"] = quantize(retval); }'
```

Which processes are receiving errors when reading from the VFS?

```
ply 'kretprobe:vfs_read if (retval < 0) { @[pid, comm, retval] = count(); }'
```

Which files are being opened, by who?

```
ply 'kprobe:do_sys_open { printf("%v(%v): %s\n", comm, uid, str(arg1)); }'
```

When sending packets, where are we coming from?

```
ply 'kprobe:dev_queue_xmit { @[stack] = count(); }'
```

From which hosts and ports are we receiving TCP resets?

```
ply 'tracepoint:tcp/tcp_receive_reset {
        printf("saddr:%v port:%v->%v\n",
                data->saddr, data->sport, data->dport);
}'
```

Source: hhttps://github.com/wkz/ply

- https://github.com/wkz/ply/tree/master/scripts
- …

2.6 Debugging eBPF

- https://docs.kernel.org/bpf/test_debug.html
- https://www.netronome.com/documents/143/UG_Getting_Started_with_eBPF_Offload.pdf
- int bpf(int cmd, union bpf_attr *attr, unsigned int size);

log_level

- 0: No debug output.
- 1: Debug information from the verifier (all instructions).
- 2: More information: add all register states after each instruction.

- llvm-objdump, llvm-mc…

bpftool

```
[mydev@fedora linux-master]$ tree tools/bpf/bpftool
tools/bpf/bpftool
├── bash-completion
│   └── bpftool
├── btf.c
├── btf_dumper.c
├── cfg.c
├── cfg.h
├── cgroup.c
├── common.c
├── Documentation
│   ├── bpftool-btf.rst
│   ├── bpftool-cgroup.rst
│   ├── bpftool-feature.rst
│   ├── bpftool-gen.rst
│   ├── bpftool-iter.rst
│   ├── bpftool-link.rst
│   ├── bpftool-map.rst
│   ├── bpftool-net.rst
│   ├── bpftool-perf.rst
│   ├── bpftool-prog.rst
│   ├── bpftool.rst
│   ├── bpftool-struct_ops.rst
│   ├── common_options.rst
│   ├── Makefile
│   └── substitutions.rst
├── feature.c
├── gen.c
├── iter.c
├── jit_disasm.c
├── json_writer.c
├── json_writer.h
├── link.c
├── main.c
├── main.h
├── Makefile
├── map.c
├── map_perf_ring.c
├── net.c
├── netlink_dumper.c
├── netlink_dumper.h
├── perf.c
├── pids.c
├── prog.c
├── skeleton
│   ├── pid_iter.bpf.c
│   ├── pid_iter.h
│   └── profiler.bpf.c
├── struct_ops.c
├── tracelog.c
├── xlated_dumper.c
└── xlated_dumper.h
```

```
# bpftool prog show
1337: sched_cls  name cls_entry  tag e202124da7c84e89
      loaded_at Mar 08/19:53  uid 0
      xlated 304B  not jited  memlock 4096B
```

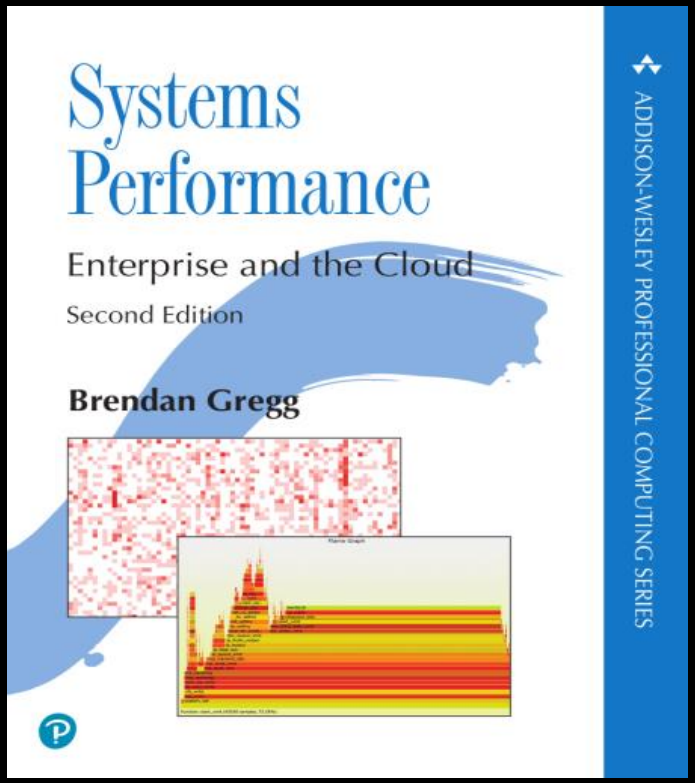
```
# bpftool prog dump xlated id 1337
   0: (71) r6 = *(u8 *)(r1 +142)
   1: (54) (u32) r6 &= (u32) 1
   2: (15) if r6 == 0x0 goto pc+7
   3: (bf) r6 = r1
   [...]
  37: (95) exit
```

```
# bpftool map
1234: array  name ch_rings  flags 0x0
      key 4B  value 4B  max_entries 7860  memlock 65536B

# bpftool map dump id 1234
key: 00 00 00 00  value: 00 00 00 00
key: 01 00 00 00  value: 00 00 00 00
key: 02 00 00 00  value: 00 00 00 00
key: 03 00 00 00  value: 00 00 00 00
[...]
Found 7860 elements
```

3) Applications

Good Resources

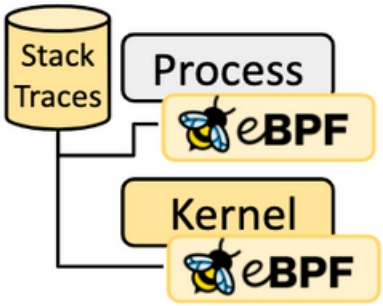
- https://ebpf.io/applications/
- https://www.brendangregg.com/systems-performance-2nd-edition-book.html



- http://cilium.readthedocs.io/en/latest/bpf/#projects-using-bpf
- https://github.com/zoidbergwill/awesome-ebpf
- ...

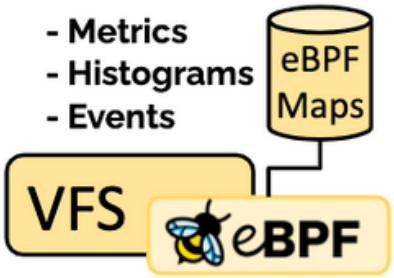
3.1 ebpf.io

- **Tracing & Profiling**



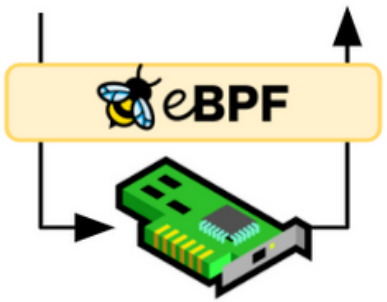
The ability to attach eBPF programs to trace points as well as kernel and user application probe points allows unprecedented visibility into the runtime behavior of applications and the system itself. By giving introspection abilities to both the application and system side, both views can be combined, allowing powerful and unique insights to troubleshoot system performance problems. Advanced statistical data structures allow to extract meaningful visibility data in an efficient manner, without requiring the export of vast amounts of sampling data as typically done by similar systems.

- **Observability & Monitoring**



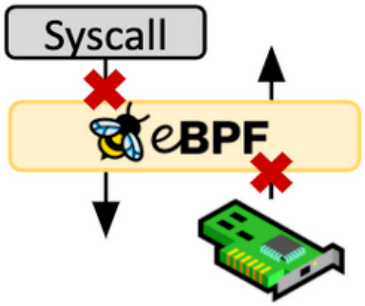
Instead of relying on static counters and gauges exposed by the operating system, eBPF enables the collection & in-kernel aggregation of custom metrics and generation of visibility events based on a wide range of possible sources. This extends the depth of visibility that can be achieved as well as reduces the overall system overhead significantly by only collecting the visibility data required and by generating histograms and similar data structures at the source of the event instead of relying on the export of samples.

- **Networking**



The combination of programmability and efficiency makes eBPF a natural fit for all packet processing requirements of networking solutions. The programmability of eBPF enables adding additional protocol parsers and easily program any forwarding logic to meet changing requirements without ever leaving the packet processing context of the Linux kernel. The efficiency provided by the JIT compiler provides execution performance close to that of natively compiled in-kernel code.

- **Security**



Building on the foundation of seeing and understanding all system calls and combining that with a packet and socket-level view of all networking operations allows for revolutionary new approaches to securing systems. While aspects of system call filtering, network-level filtering, and process context tracing have typically been handled by completely independent systems, eBPF allows for combining the visibility and control of all aspects to create security systems operating on more context with better level of control.

3.2 eBPF-assisted Infrastructure

- From our perspective:
  https://github.com/XianBeiTuoBaFeng2015/MySlides/blob/master/Conf/2021/K%2BSummit2021__Revisiting%20eBPF-centric%20New%20Design%20for%20HCI%20and%20Edge%20Computing__FengLi-updated20220405p.pdf

IV. Networking

- Emerging Networking Technologies
- eBPF for Cloud-native Networking
- eBPF/XDP Hardware Offload to SmartNIC/DPU
- P4 and eBPF

V. Messaging & RPC

- Lightweight RPC
- Hardware-accelerated RPC
- eBPF-inspired Messaging and RPC

VI. HPC

- Standards and Frameworks
- New Runtime for HPC

VII. Storage

- Background
- eBPF-powered Userland Filesystems
- eBPF in Computational Storage
- eBPF-based In-kernel Storage

VIII. Distributed AI

- Data Processing
- Project Ray

IX. Blockchain

- eBPF in Blockchain

X. Security

- Overview
- eBPF-based Linux System Security
- eBPF in Firewall
- eBPF for Cloud-native Security

XI. System Debugging, Tuning, and Monitoring

- eBPF-based Unified Debugger
- eBPF in System and Application Profiling
- eBPF for System Observability and Monitoring

XII. Cloud-native

- eBPF for Containers
- eBPF in Kubernetes Ecosystem
- eBPF in Microservices
- Project Cilium
- Project BumbleBee
- Project Kindling

XIII. 5G-6G

- 5G & 6G
- XDP for 5G UPF

XIV. xBPF

- Overview
- Femto-Containers

...

IV. The future of eBPF

- http://vger.kernel.org/bpfconf2022_material/lsfmmbpf2022-bpf-wip-roadmap.pdf
- ...

1) Programming
1.1 Moving to modern C

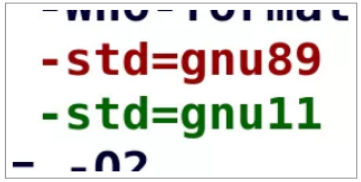
- Linux Kernel Moving Ahead With Going From C89 To C11 Code

Written by Michael Larabel in Linux Kernel on 28 February 2022 at 07:06 AM EST. 69 Comments

It looks like for the Linux 5.18 kernel cycle coming up it could begin allowing modern C11 code to be accepted rather than the current Linux kernel codebase being limited to the C89 standard.

Following mailing list discussions, Linus Torvalds entertained the idea of bumping the C version target from C89 up to C99. But it turns out with the current minimum version compiler requirements of the kernel and the condition of the current code, they can actually begin building the kernel with C11 in mind.

Thanks to Linux 5.15 raising the compiler requirement to GCC 5.1 and other recent improvements to the code-base, they can now begin safely building the Linux kernel using C11/GNU11 for its accepted C version.



Allowing modern C code into the Linux kernel!

This morning Arnd Bergmann sent out the new patch allowing the Linux kernel to default to "*-std=gnu11*" in specifying the GNU dialect of C11. Thus moving forward the kernel will allow usage of nice C99/C11 features rather than being limited to C89. As this change already has the blessing of Linus Torvalds, it will likely go forward in the next kernel merge window assuming no fundamental issues are uncovered.

Source: https://www.phoronix.com/news/Linux-Kernel-C89-To-C11

- **https://lwn.net/Articles/885941/**
- **https://en.wikipedia.org/wiki/C11_(C_standard_revision)**

1.2 Rust for Linux

1.2.1 What's new in Rust

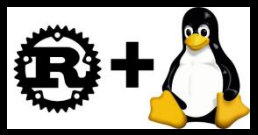
- https://foundation.rust-lang.org/

- https://www.infoworld.com/article/3267624/whats-new-in-the-rust-language.html
- https://github.com/RalfJung/minirust
  A precise specification for "Rust lite / MIR plus".
- https://github.com/rust-lang/miri/
  An interpreter for Rust's mid-level intermediate representation.
- https://lwn.net/Articles/900721/ //Rust frontend approved for GCC
- https://foundation.rust-lang.org/news/2022-09-13-rust-foundation-establishes-security-team/
- https://github.com/awesome-rust-cloud-native/awesome-rust-cloud-native

Roadmap

- https://blog.rust-lang.org/inside-rust/2022/04/04/lang-roadmap-2024.html
- https://blog.rust-lang.org/inside-rust/2022/02/22/compiler-team-ambitions-2022.html
- https://www.ncameron.org/blog/ten-challenges-for-rust/

1.2.2 Rust heads into Linux Kernel



- **What's happening!**

  https://lwn.net/Articles/889924/ //Rustaceans at the border
  https://lwn.net/Articles/853423/ //Rust heads into the kernel?
  https://lwn.net/Articles/852704/ //Rust in the Linux kernel
  https://lwn.net/Articles/849849/ //Rust support hits linux-next
  https://lwn.net/Articles/829858/ //Supporting Linux kernel development in Rust
  …

- **https://github.com/Rust-for-Linux**

  The goal of this project is to add support for the Rust language to the Linux kernel. This repository contains the work that will be eventually submitted for review to the LKML.

  Feel free to contribute! To start, take a look at `Documentation/rust`.

- **Rust for Linux patch 9**

  https://www.phoronix.com/news/Rust-For-Linux-v9-Patches

  Earlier this week saw the Rust for Linux v8 patches posted that introduced a number of new abstractions and expanding the Rust programming language integration to more areas of the kernel. Those patches amounted to 43.6k lines of new code while "Rust for Linux v9" was posted today and comes in at just 12.5k lines of new code.

  Rust for Linux v9 is significantly smaller than the prior patches due to removing a lot of extra features and integration. The hope is now to take a more initial minimal route with the Rust for Linux integration until that initial mainlining and then from there can build things up with the enhanced integration and allowing more involved review/feedback of the various abstractions and subsystem-specific patches.

  https://lore.kernel.org/lkml/20220805154231.31257-1-ojeda@kernel.org/

  ```
  89 files changed, 12548 insertions(+), 51 deletions(-)
  ```

- **https://www.phoronix.com/news/Rust-For-Linux-5.20-Possible (Linux 6.0)**

- **Memory Safety for the World's Largest Software Project**

  https://lwn.net/Articles/899164/

  Miguel Ojeda has posted an update on the Rust-for-Linux project.

  > This second year since the RFC we are looking forward to several milestones which hopefully we will achieve:
  >
  > - More users or use cases inside the kernel, including example drivers – this is pretty important to get merged into the kernel.
  > - Splitting the kernel crate and managing dependencies to allow better development.
  > - Extending the current integration of the kernel documentation, testing and other tools.
  > - Getting more subsystem maintainers, companies and researchers involved.
  > - Seeing most of the remaining Rust features stabilized.
  > - Possibly being able to start compiling the Rust code in the kernel with GCC.
  > - And, of course, getting merged into the mainline kernel, which should make everything else easier!

- **https://www.memorysafety.org/blog/memory-safety-in-linux-kernel/**
- **r-linux**

  https://github.com/bus1/r-linux **//Capability-based Linux Runtime**

  The r-linux project provides direct access to the application programming interfaces of the linux kernel. This includes direct unprotected accessors to the kernel API, as well as rustified traits and functions to access the kernel API in a safe, capability-based way.

- **Kernel Driver with Rust**

  https://www.phoronix.com/news/LPC-2022-Rust-Linux

  https://not-matthias.github.io/posts/kernel-driver-with-rust-2022/

  …

- https://www.theregister.com/2022/09/16/rust_in_the_linux_kernel/
- https://blog.rust-lang.org/2022/08/01/Increasing-glibc-kernel-requirements.html
- https://www.linuxfoundation.org/webinars/writing-linux-kernel-modules-in-rust

---

- https://kangrejos.com/
- …

Get Rust into Linux kernel 6.1?

- https://www.zdnet.com/article/linus-torvalds-rust-may-make-it-into-the-next-linux-kernel-after-all/

1.2.3 Rust for eBPF

Overview

- https://kbknapp.dev/ebpf-part-i/
- https://kbknapp.dev/ebpf-part-ii/
- https://kbknapp.dev/ebpf-part-iii/
- https://kbknapp.dev/ebpf-part-iv/
- …

1.2.3.1 Aya

- https://github.com/aya-rs/aya
  An eBPF library for the Rust programming language, built with a focus on developer experience and operability.
- **Features**

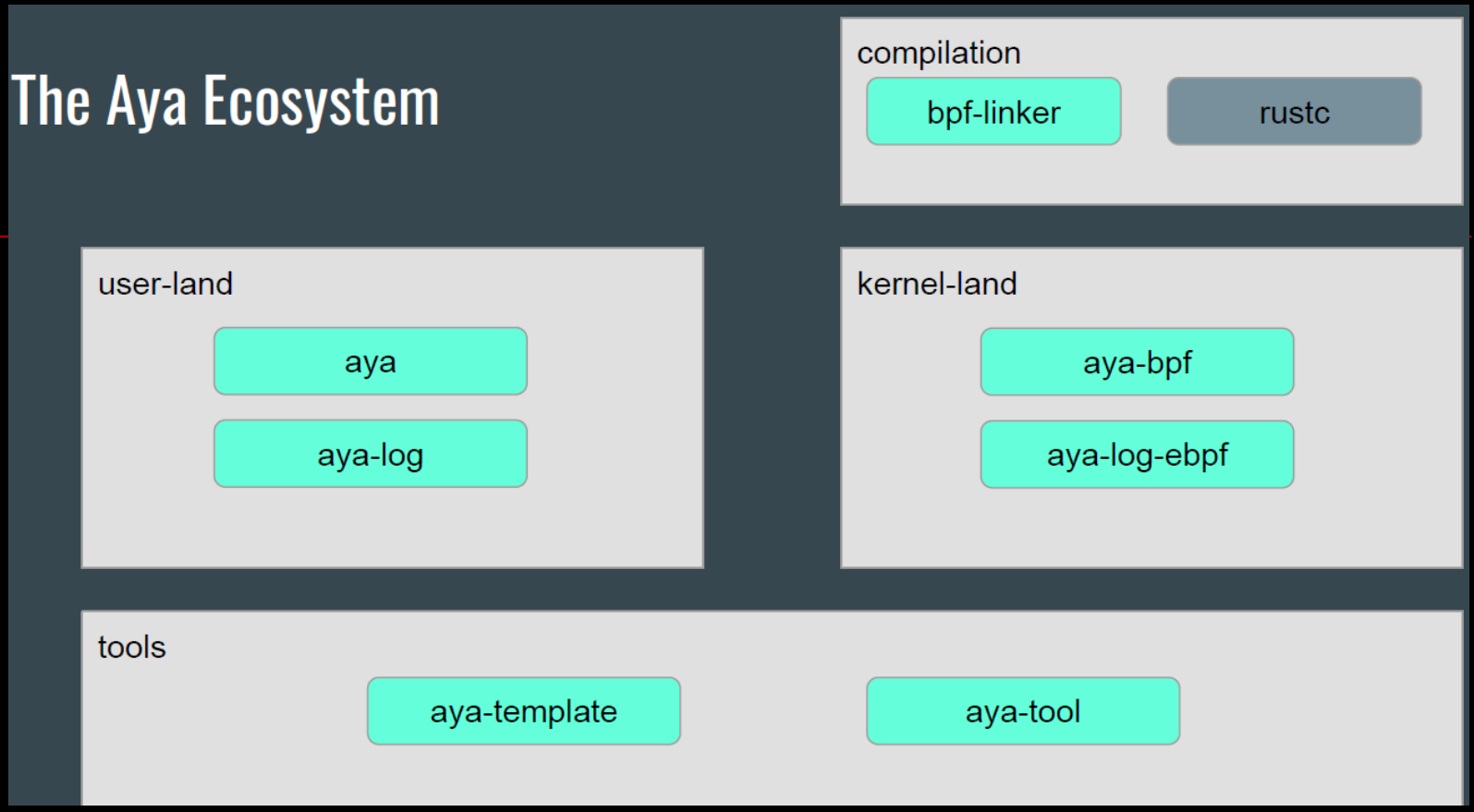
Aya is an eBPF library built with a focus on operability and developer experience. It does not rely on libbpf nor bcc - it's built from the ground up purely in Rust, using only the libc crate to execute syscalls. With BTF support and when linked with musl, it offers a true compile once, run everywhere solution, where a single self-contained binary can be deployed on many linux distributions and kernel versions.

Some of the major features provided include:

- Support for the **BPF Type Format** (BTF), which is transparently enabled when supported by the target kernel. This allows eBPF programs compiled against one kernel version to run on different kernel versions without the need to recompile.
- Support for function call relocation and global data maps, which allows eBPF programs to make **function calls** and use **global variables and initializers**.
- **Async support** with both tokio and async-std.
- Easy to deploy and fast to build: aya doesn't require a kernel build or compiled headers, and not even a C toolchain; a release build completes in a matter of seconds.

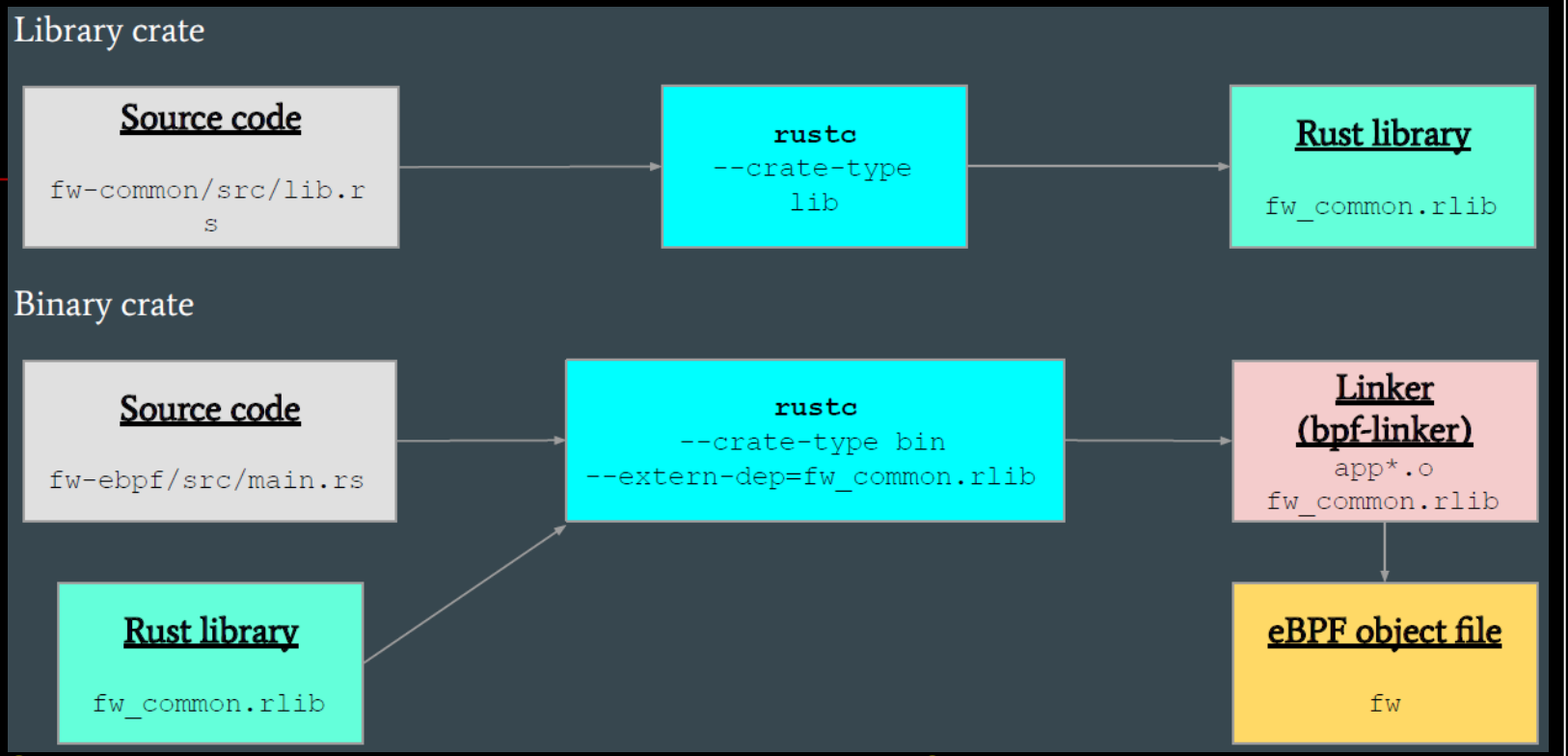
- https://lwn.net/Articles/859784/ //Aya: writing BPF in Rust
- https://aya-rs.dev/book/
- https://github.com/aya-rs/bpf-linker
- https://github.com/aya-rs/awesome-aya
- https://deepfence.io/aya-your-trusty-ebpf-companion/

The Aya Ecosystem



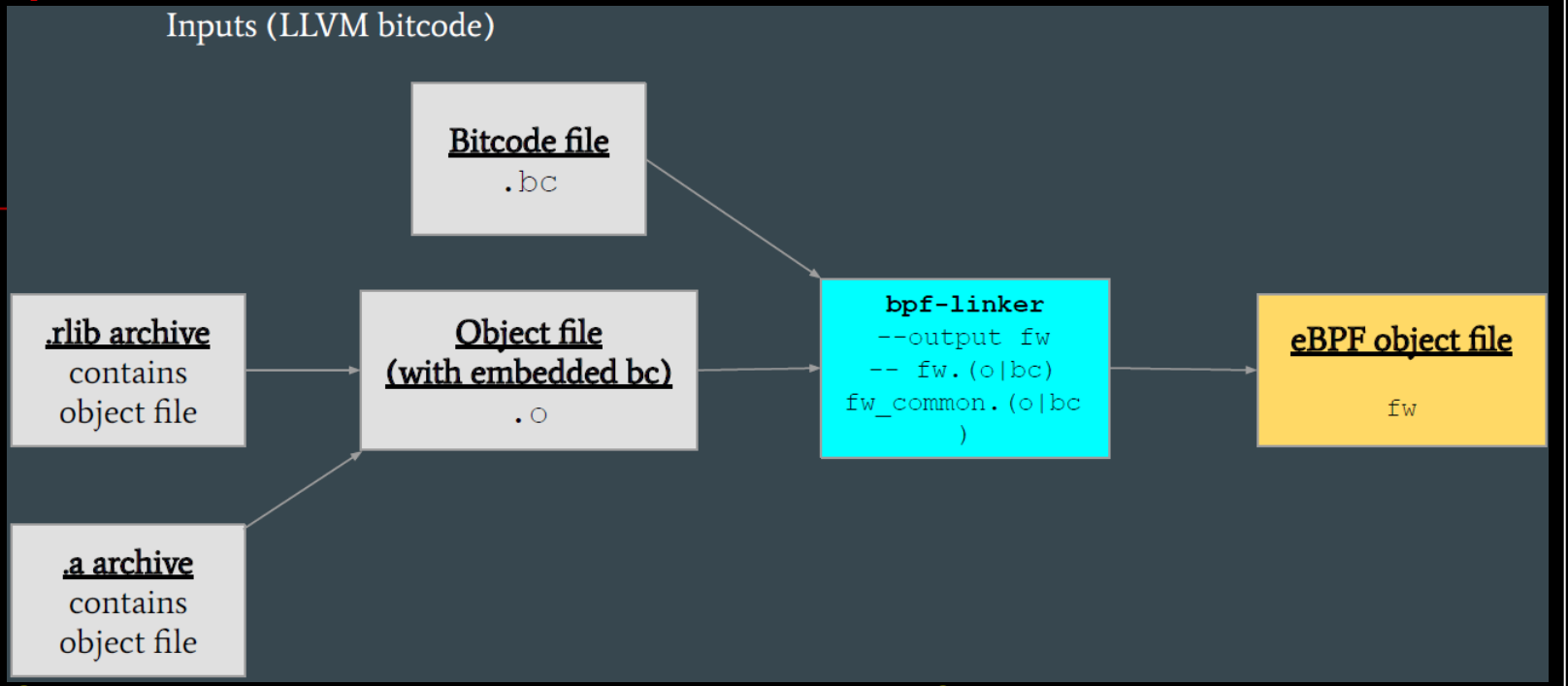
Source: “Rust in the Kernel (via eBPF)”, Dave Tucker etc, LPC 2022.

Rust -> eBPF compilation



Source: "Rust in the Kernel (via eBPF)", Dave Tucker etc, LPC 2022.

bpf-linker



Source: "Rust in the Kernel (via eBPF)", Dave Tucker etc, LPC 2022.

1.3 Technology trends of LLVM

1.3.1 MLIR

- https://mlir.llvm.org/

  Multi-Level Intermediate Representation

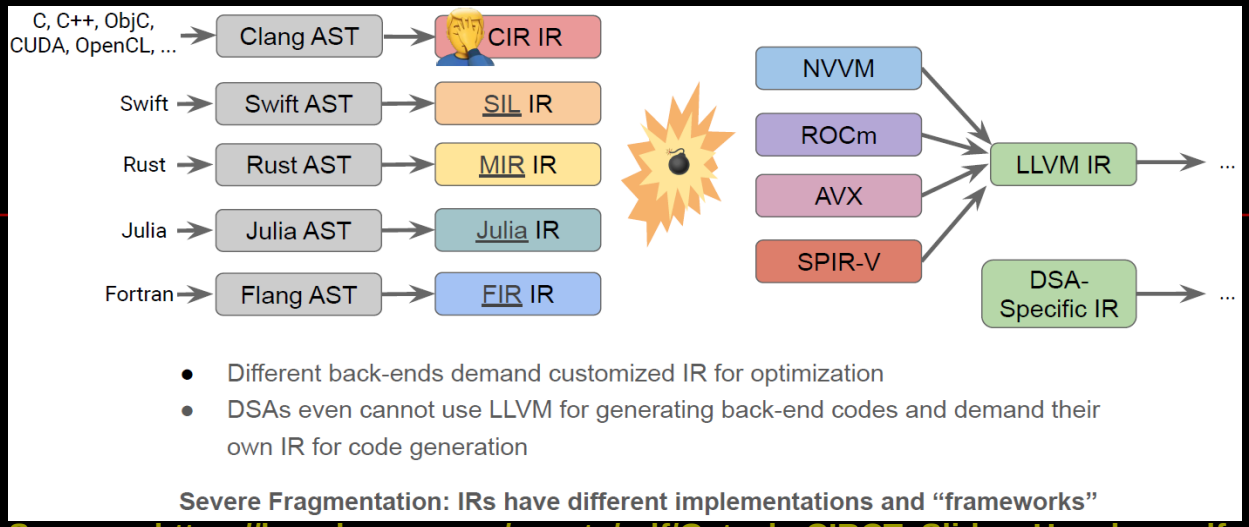
The MLIR project is a novel approach to building reusable and extensible compiler infrastructure. MLIR aims to address software fragmentation, improve compilation for heterogeneous hardware, significantly reduce the cost of building domain specific compilers, and aid in connecting existing compilers together.

- Motivation

MLIR is intended to be a hybrid IR which can support multiple different requirements in a unified infrastructure. For example, this includes:

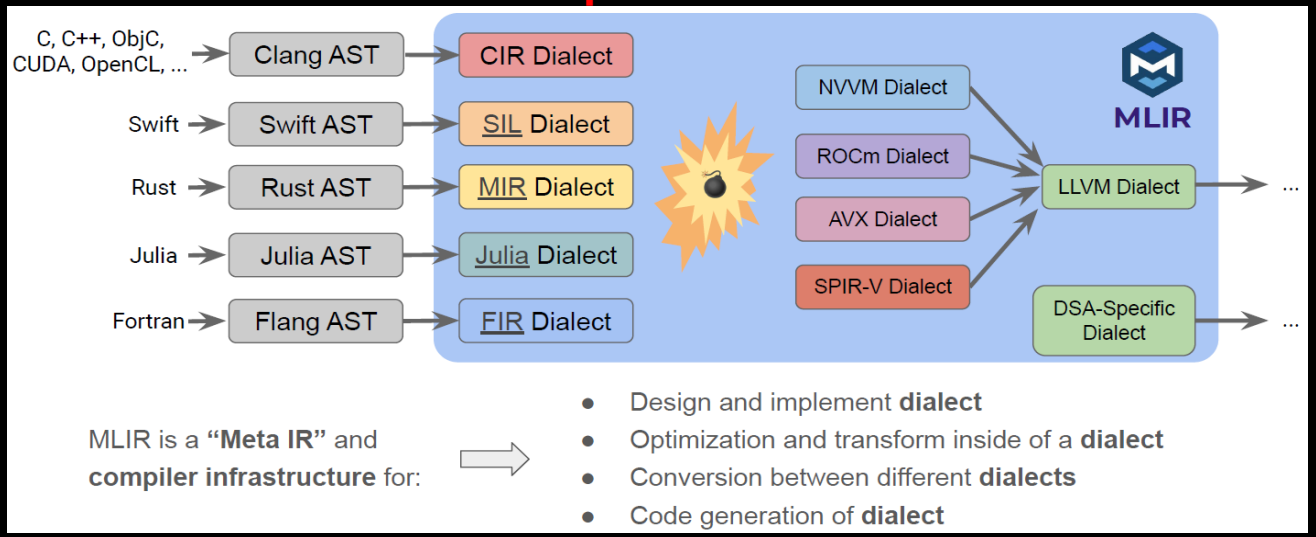
- The ability to represent dataflow graphs (such as in TensorFlow), including dynamic shapes, the user-extensible op ecosystem, TensorFlow variables, etc.
- Optimizations and transformations typically done on such graphs (e.g. in Grappler).
- Ability to host high-performance-computing-style loop optimizations across kernels (fusion, loop interchange, tiling, etc.), and to transform memory layouts of data.
- Code generation “lowering” transformations such as DMA insertion, explicit cache management, memory tiling, and vectorization for 1D and 2D register architectures.
- Ability to represent target-specific operations, e.g. accelerator-specific high-level operations.
- Quantization and other graph transformations done on a Deep-Learning graph.
- Polyhedral primitives.
- Hardware Synthesis Tools / HLS.

From LLVM to MLIR



Source: https://hanchenye.com/assets/pdf/Gatech_CIRCT_Slides_Hanchen.pdf

MLIR: "Meta IR" and Compiler Infrastructure

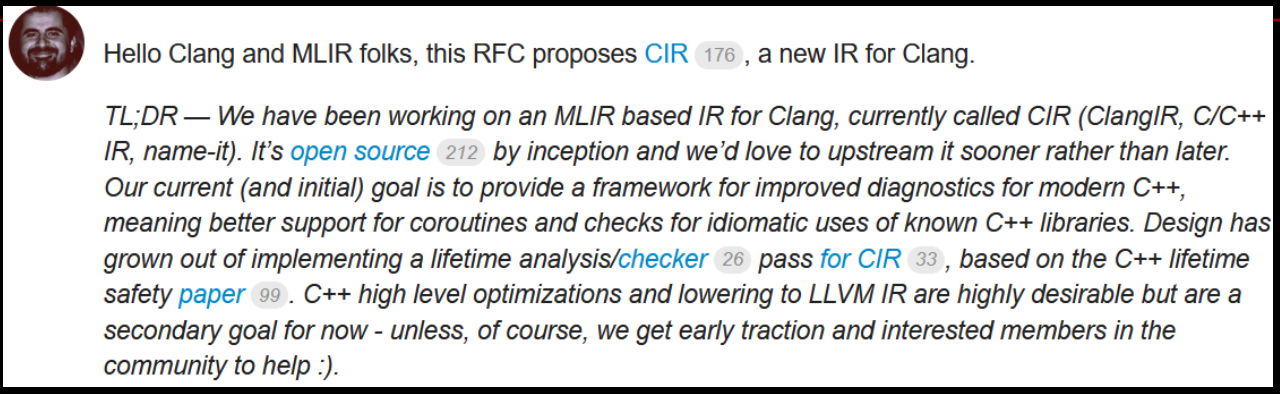


Source: https://hanchenye.com/assets/pdf/Gatech_CIRCT_Slides_Hanchen.pdf

1.3.2 ClangIR (CIR)

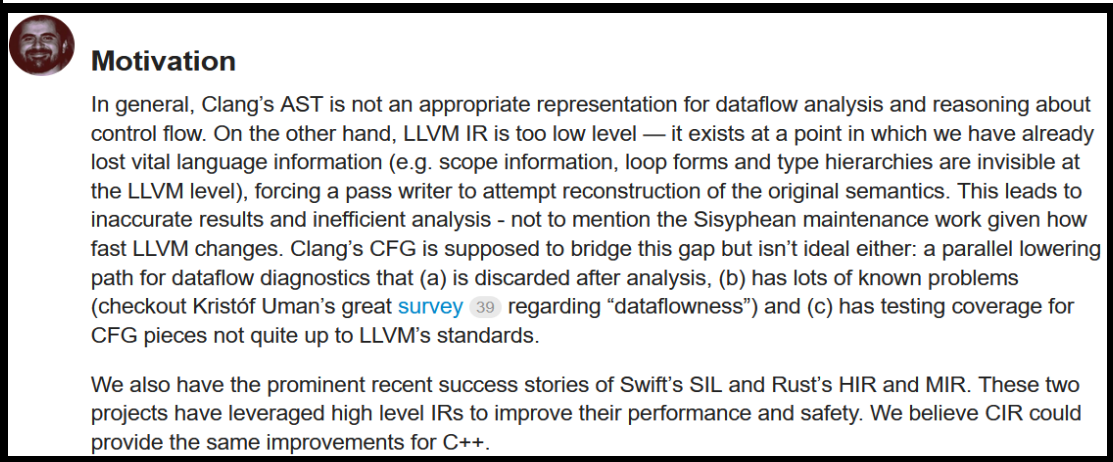
- https://clangir.org/

  A new IR for Clang.
- https://discourse.llvm.org/t/rfc-an-mlir-based-clang-ir-cir/63319



Hello Clang and MLIR folks, this RFC proposes CIR 176, a new IR for Clang.

TL;DR — We have been working on an MLIR based IR for Clang, currently called CIR (ClangIR, C/C++ IR, name-it). It's open source 212 by inception and we'd love to upstream it sooner rather than later. Our current (and initial) goal is to provide a framework for improved diagnostics for modern C++, meaning better support for coroutines and checks for idiomatic uses of known C++ libraries. Design has grown out of implementing a lifetime analysis/checker 26 pass for CIR 33, based on the C++ lifetime safety paper 99. C++ high level optimizations and lowering to LLVM IR are highly desirable but are a secondary goal for now - unless, of course, we get early traction and interested members in the community to help :).



Motivation

In general, Clang's AST is not an appropriate representation for dataflow analysis and reasoning about control flow. On the other hand, LLVM IR is too low level — it exists at a point in which we have already lost vital language information (e.g. scope information, loop forms and type hierarchies are invisible at the LLVM level), forcing a pass writer to attempt reconstruction of the original semantics. This leads to inaccurate results and inefficient analysis - not to mention the Sisyphean maintenance work given how fast LLVM changes. Clang's CFG is supposed to bridge this gap but isn't ideal either: a parallel lowering path for dataflow diagnostics that (a) is discarded after analysis, (b) has lots of known problems (checkout Kristóf Uman's great survey 39 regarding "dataflowness") and (c) has testing coverage for CFG pieces not quite up to LLVM's standards.

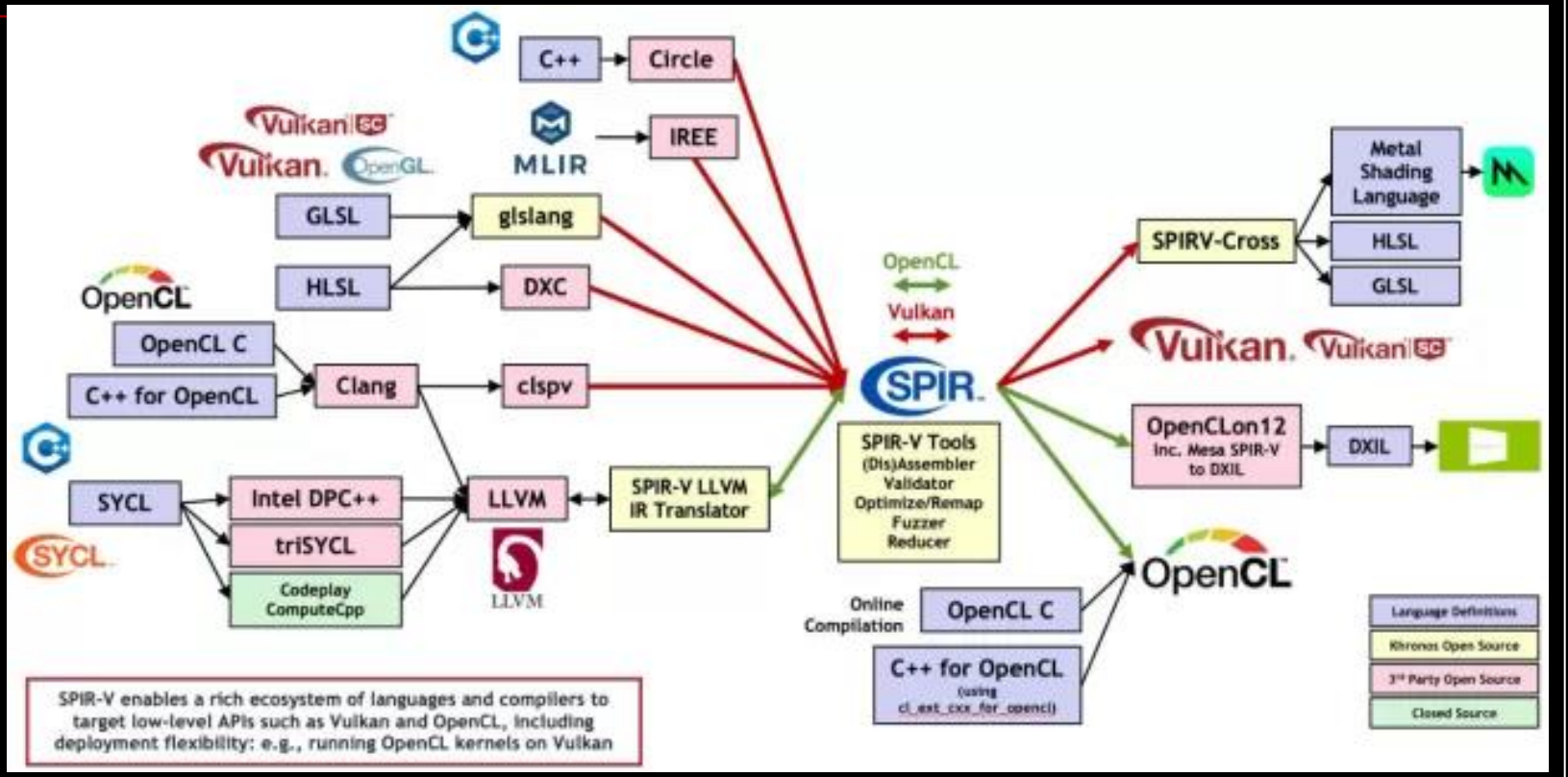
We also have the prominent recent success stories of Swift's SIL and Rust's HIR and MIR. These two projects have leveraged high level IRs to improve their performance and safety. We believe CIR could provide the same improvements for C++.

- https://llvm.github.io/clangir/
- https://github.com/llvm/clangir
- …

1.3.3 Initial SPIR-V Backend Code Lands In LLVM 15

- https://www.phoronix.com/news/LLVM-15-SPIR-V-Backend

  SPIR-V is at the heart of standards/software and with a mainline LLVM back-end can ultimately open it up to even more.



- https://github.com/llvm/llvm-project/commit/7fd4622d4801cc14823ecde678d55b6f3a106eb9
- https://github.com/KhronosGroup/LLVM-SPIRV-Backend

1.4 eBPF on RISC-V
1.4.1 Src

-
```
[mydev@fedora linux-master]$ tree arch/riscv/net/
arch/riscv/net/
├── bpf_jit_comp32.c
├── bpf_jit_comp64.c
├── bpf_jit_core.c
├── bpf_jit.h
└── Makefile
```

```
[mydev@fedora linux-master]$ grep -ir "BPF" arch/riscv
arch/riscv/configs/defconfig:CONFIG_BPF_SYSCALL=y
arch/riscv/configs/defconfig:CONFIG_CGROUP_BPF=y
arch/riscv/configs/rv32_defconfig:CONFIG_BPF_SYSCALL=y
arch/riscv/configs/rv32_defconfig:CONFIG_CGROUP_BPF=y
arch/riscv/include/asm/asm-extable.h:#define EX_TYPE_BPF                     2
arch/riscv/include/asm/extable.h:#if defined(CONFIG_BPF_JIT) && defined(CONFIG_ARCH_RV64I)
arch/riscv/include/asm/extable.h:bool ex_handler_bpf(const struct exception_table_entry *ex, struct pt_regs *regs);
arch/riscv/include/asm/extable.h:ex_handler_bpf(const struct exception_table_entry *ex,
arch/riscv/include/asm/perf_event.h:#define perf_arch_bpf_user_pt_regs(regs) (struct user_regs_struct *)regs
arch/riscv/include/asm/pgtable.h:/* Leave 2GB for kernel and BPF at the end of the address space */
arch/riscv/include/asm/pgtable.h:#define BPF_JIT_REGION_SIZE    (SZ_128M)
arch/riscv/include/asm/pgtable.h:#define BPF_JIT_REGION_START   (BPF_JIT_REGION_END - BPF_JIT_REGION_SIZE)
arch/riscv/include/asm/pgtable.h:#define BPF_JIT_REGION_END     (MODULES_END)
arch/riscv/include/asm/pgtable.h:#define BPF_JIT_REGION_START   (PAGE_OFFSET - BPF_JIT_REGION_SIZE)
arch/riscv/include/asm/pgtable.h:#define BPF_JIT_REGION_END     (VMALLOC_END)
arch/riscv/include/uapi/asm/bpf_perf_event.h:#ifndef _UAPI__ASM_BPF_PERF_EVENT_H__
arch/riscv/include/uapi/asm/bpf_perf_event.h:#define _UAPI__ASM_BPF_PERF_EVENT_H__
arch/riscv/include/uapi/asm/bpf_perf_event.h:typedef struct user_regs_struct bpf_user_pt_regs_t;
arch/riscv/include/uapi/asm/bpf_perf_event.h:#endif /* _UAPI__ASM_BPF_PERF_EVENT_H__ */
arch/riscv/mm/ptdump.c: {0, "Modules/BPF mapping"},
arch/riscv/mm/extable.c:        case EX_TYPE_BPF:
arch/riscv/mm/extable.c:                return ex_handler_bpf(ex, regs);
arch/riscv/mm/init.c:    * space is occupied by the modules/BPF/kernel mappings which reduces
arch/riscv/mm/kasan_init.c:     /* Populate kernel, BPF, modules mapping */
...
```

- **Maturing at a fast pace…**

1.5 More DSLs for eBPF programming

DSL

- https://en.wikipedia.org/wiki/Domain-specific_language

2) Deeply integration into the Cloud-native

Cloud-native

- https://en.wikipedia.org/wiki/Cloud_native_computing

Cloud native computing is an approach in software development that utilizes cloud computing to "build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds".[1] Technologies such as containers, microservices, serverless functions and immutable infrastructure, deployed via declarative code are common elements of this architectural style.[2][3]

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

Frequently, cloud-native applications are built as a set of microservices that run in Docker containers, and may be orchestrated in Kubernetes and managed and deployed using DevOps and Git CI workflows[4] (although there is a large amount of competing open source that supports cloud-native development). The advantage of using Docker containers is the ability to package all software needed to execute into one executable package. The container runs in a virtualized environment, which isolates the contained application from its environment.[2]

- https://landscape.cncf.io/
- https://www.cloud-native.wiki/
- https://en.wikipedia.org/wiki/OS-level_virtualization
- https://en.wikipedia.org/wiki/Kubernetes
- …

2.1 Project Cilium

2.1.1 Overview

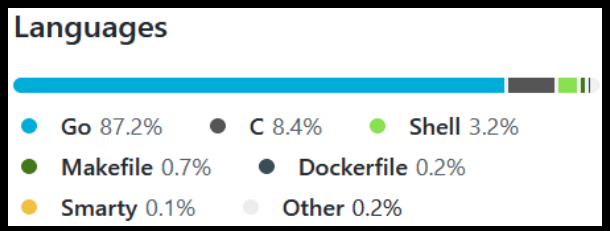
- https://cilium.io/
- https://github.com/cilium/cilium

eBPF-based Networking, Security, and Observability.

Cilium is a networking, observability, and security solution with an eBPF-based dataplane. It provides a simple flat Layer 3 network with the ability to span multiple clusters in either a native routing or overlay mode. It is L7-protocol aware and can enforce network policies on L3-L7 using an identity based security model that is decoupled from network addressing.

Cilium implements distributed load balancing for traffic between pods and to external services, and is able to fully replace kube-proxy, using efficient hash tables in eBPF allowing for almost unlimited scale. It also supports advanced functionality like integrated ingress and egress gateway, bandwidth management and service mesh, and provides deep network and security visibility and monitoring.

A new Linux kernel technology called eBPF is at the foundation of Cilium. It supports dynamic insertion of eBPF bytecode into the Linux kernel at various integration points such as: network IO, application sockets, and tracepoints to implement security, networking and visibility logic. eBPF is highly efficient and flexible. To learn more about eBPF, visit eBPF.io.



- https://docs.cilium.io/en/latest/

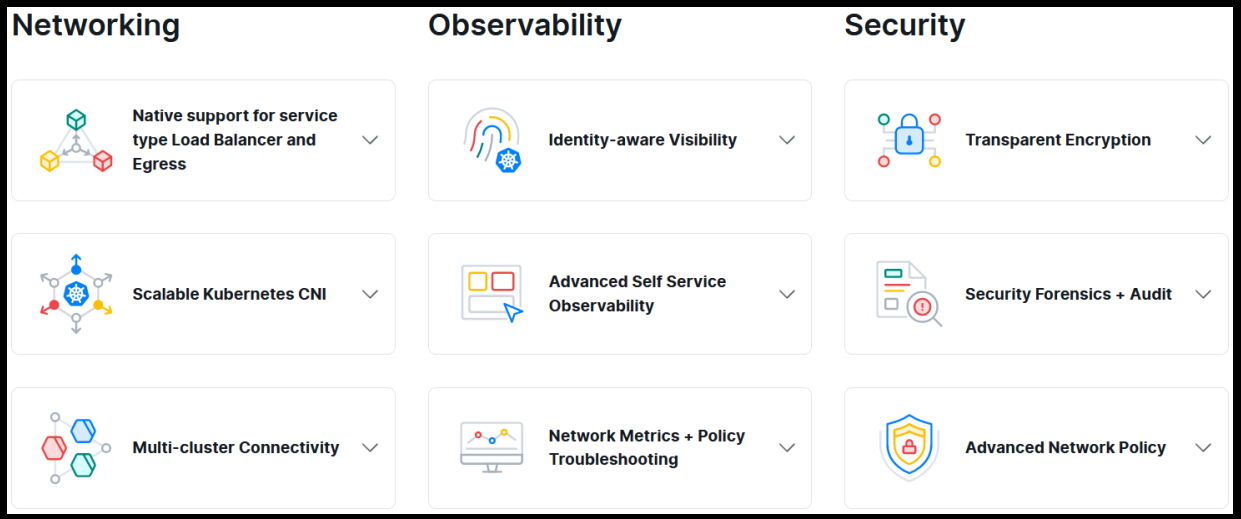
What is Cilium?

Cilium is an open source project to provide networking, security, and observability for cloud native environments such as Kubernetes clusters and other container orchestration platforms.

At the foundation of Cilium is a new Linux kernel technology called eBPF, which enables the dynamic insertion of powerful security, visibility, and networking control logic into the Linux kernel. eBPF is used to provide high-performance networking, multi-cluster and multi-cloud capabilities, advanced load balancing, transparent encryption, extensive network security capabilities, transparent observability, and much more.

Source: https://cilium.io/get-started/

- **Features**

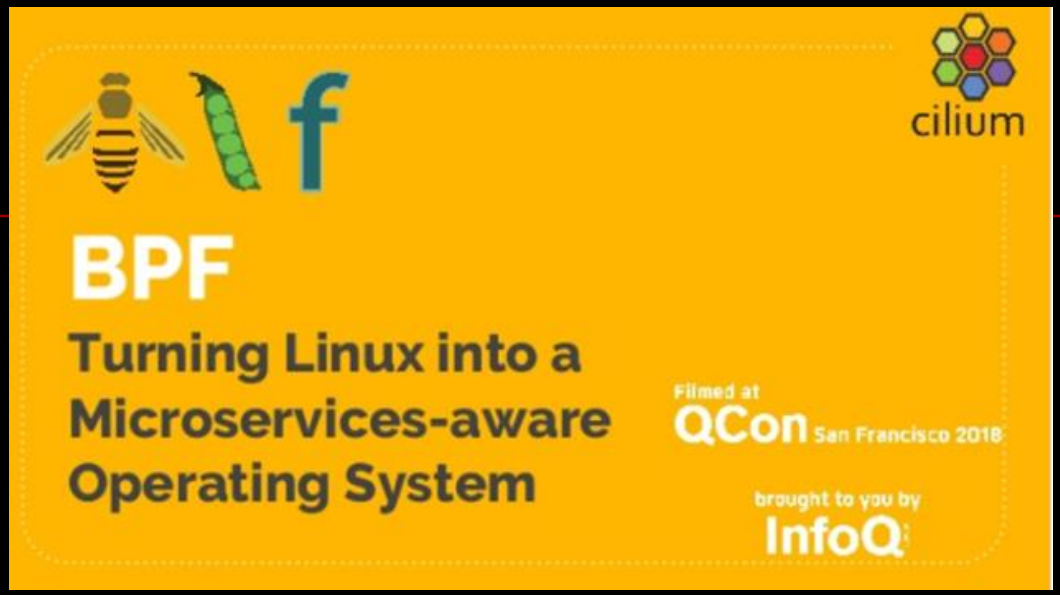


Source: https://cilium.io/

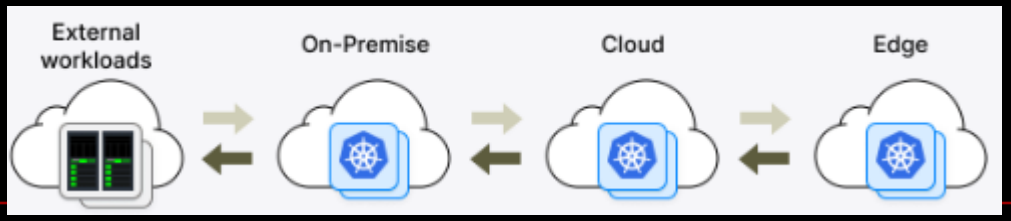
- **Roadmap**

https://docs.cilium.io/en/latest/community/roadmap/

- **Cilium's ambition**



Architecture & Design



Service Mesh

RESILIENCE AUTHENTICATION TRAFFIC MANAGEMENT

cilium Service Mesh — Ingress — Gateway API — spiffe

Observability

METRICS TRACING SERVICE MAP LOGS

cilium Hubble — SIEM — JSON — fluentd — Grafana — OpenTelemetry

Networking

ENCRYPTION LOAD-BALANCING NETWORK POLICY NETWORKING

cilium CNI

IPsec
Wireguard

Kubernetes
DSR • Maglev

DNS • L7
L3/L4

IPv4 • IPv6 • Overlay • BGP
Egress Gateway • Multi-Cluster

Runtime Security

OBSERVABILITY
ENFORCEMENT

Tetragon

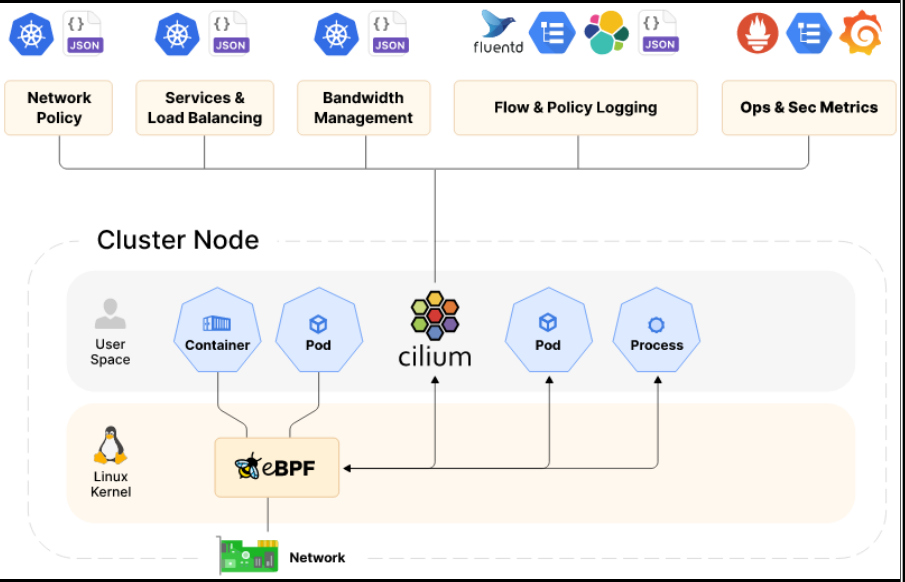
SIEM JSON fluentd

Kubernetes Container VM Metal

aws Google Cloud Azure Alibaba Cloud Red Hat OpenShift vmware

Source: https://github.com/cilium/cilium/blob/master/Documentation/images/cilium-overview.png

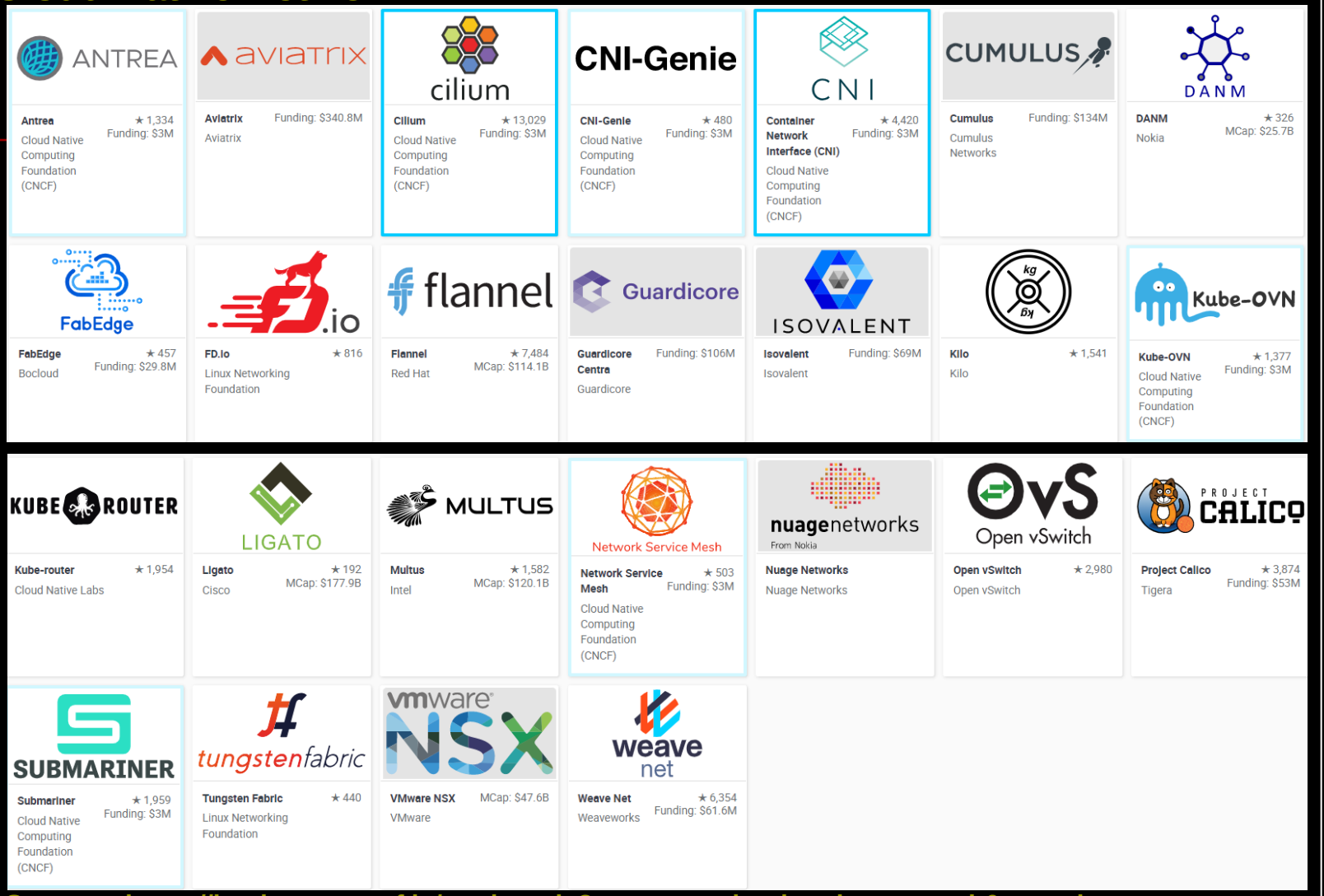
Cilium consists of an agent running on all cluster nodes and servers in your environment. It provides networking, security, and observability to the workloads running on that node. Workloads can be containerized or running natively on the system.



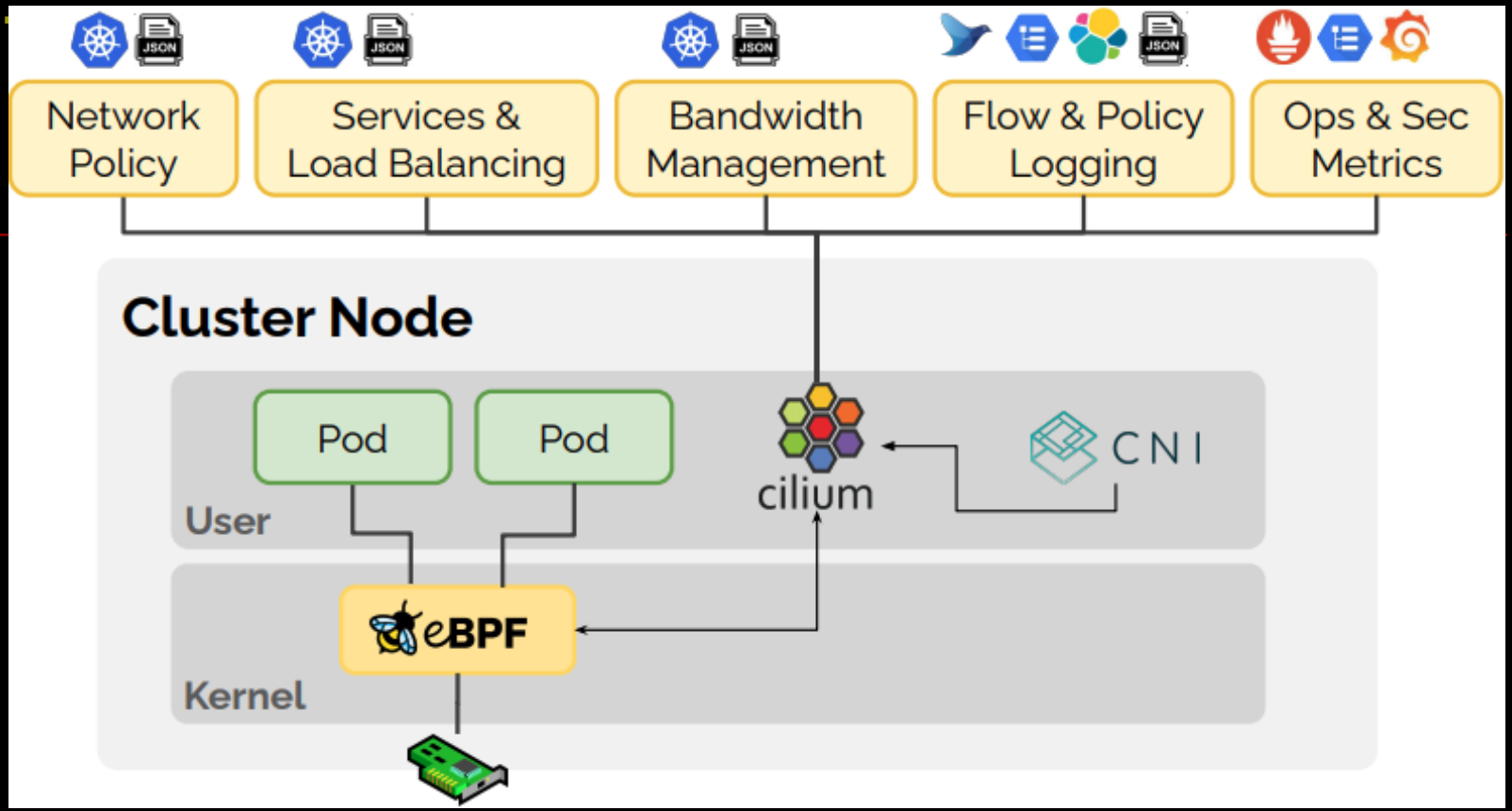
Source: https://cilium.io/get-started/

Cilium as the CNI

- Cloud Native Network



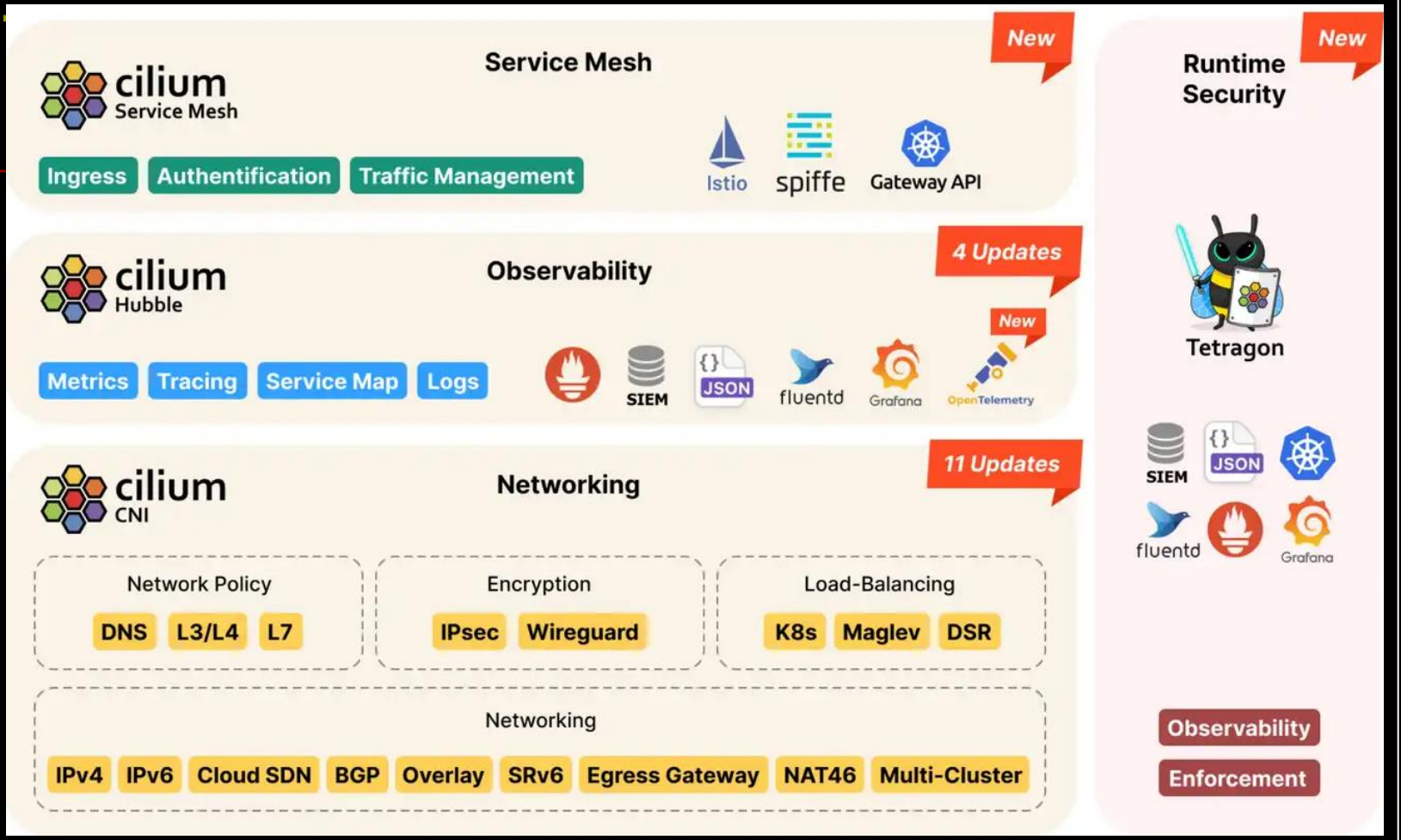
Source: https://landscape.cncf.io/card-mode?category=cloud-native-network&grouping=category



Source: https://github.com/cilium/cilium/blob/master/Documentation/images/cilium_overview.png

- https://docs.cilium.io/en/latest/operations/performance/benchmark/
- https://www.spectrocloud.com/blog/getting-started-with-cilium-for-kubernetes-networking-and-observability/
- https://dev.to/otomato_io/cilium-ebpf-powered-cni-a-nos-solution-for-modern-clouds-1hl1
- ...

Cilium 1.12



Source: https://isovalent.com/blog/post/cilium-release-112/

New Features at a Glance:
(https://isovalent.com/blog/post/cilium-release-112/)

Service Mesh & Ingress:

- **Integrated Ingress Controller**: A fully compliant Kubernetes Ingress controller embedded into Cilium. Additional annotations are supported for more advanced use cases. (More details)
- **Sidecar-free datapath option**: A new datapath option for Cilium Service Mesh as an alternative to the Istio integration allowing to run a service mesh without sidecars. (More details)
- **Envoy CRD**: A new Kubernetes CRD making the full power of Envoy available whereever Cilium runs. You can express your requirements in an Envoy Configuration and inject it anywhere in the network. (More details)
- **Gateway API**: The work on supporting Gateway API has started. If you are interested in contributing, reach out on Slack.

Cluster Mesh:

- **Topology-aware routing and service affinity**: With a single line of YAML annotation, services can be configured to prefer endpoints in the local or remote cluster, if endpoints in multiple clusters are available. (More details)
- **Simplified cluster connections with Helm**: New simplified user experience to connect Kubernetes clusters together using Cilium and Helm. (More details)
- **Lightweight Multi-Cluster for External Workloads**: Cluster Mesh now supports special lightweight remote clusters. This allows running lightweight clusters for use by external workloads. (More details)

External Workload Improvements:

- **Egress Gateway promoted to Stable**: Cilium enables users to route selected cluster-external connections through specific Gateway nodes, masquerading them with predictable IP addresses to allow integration with traditional firewalls that require static IP addresses. A new `CiliumEgressGatewayPolicy` CRD improves the selection of the Gateway node and its egress interface. Additional routing logic ensures compatibility with ENI interfaces. (More details)
- **Improved BGP control plane**: IPv6 support has been added to the BGP control plane. By leveraging a new feature-rich BGP engine, Cilium can now set up IPv6 peering sessions and advertise BGP IPv6 Pod CIDRs. (More details)
- **VTEP support**: This new integration allows Cilium to peer with VXLAN Tunnel Endpoint devices in on-prem environments. (More details)

Security:

- **Running Cilium as unprivileged Pod**: You can now run Cilium as an unprivileged container/Pod to reduce the attack surface of a Cilium installation. (More details)
- **Reduction in required Kubernetes privileges**: The required Kubernetes privileges have been greatly reduced to the least needed for Cilium to operate. (More details)
- **Network policies for ICMP**: Cilium users can now allow a subset of ICMP traffic on egress and ingress, with the usual `CiliumNetworkPolicies` and `CiliumClusterwideNetworkPolicies`. (More details)

Load-Balancing:

- **L7 Load-balancing:** With the addition of Ingress support, Cilium has become capable of performing L7 load-balancing. (More details)
- **NAT46/64 Support for Load Balancer**: Cilium L4 load-balancer (L4LB) now supports NAT46 and NAT64 for services. This allows exposing an IPv6-only Pod via an IPv4 service IP or vice versa. This is particularly useful to load-balance IPv4 client traffic at the edge to IPv6-only clusters. (More details)
- **Quarantining Service backends**: A new API to quarantine service backends that are unreachable, and unable to do load-balancing. The latter can be utilized by health checker components to drain traffic from unstable backends, or backends that should be placed into maintenance mode. (More details)
- **Improved Multi-Homing for Load Balancer**: Cilium's datapath is extended to support multiple native network devices and multiple paths. (More details)

User Experience:

- **Automatic Helm Value Discovery**: Cilium CLI is now capable of automatically discovering cluster types and generating the ideal helm config values for the discovered cluster type. (More details)
- **AKS BYOCNI support**: Cilium and the Cilium CLI now support AKS clusters created with the new Bring-Your-Own Container Network Interface (BYOCNI) mode. (More details)
- **Improved chaining mode support**: Improved integration between Cilium and cloud CNI plugins for Azure and AWS in chaining mode. (More details)
- **Better troubleshooting with Hubble CLI**: Many improvements to the Hubble CLI including a better indication of whether a particular connection has been allowed or denied. (More details)

Networking:

- **BBR congestion control for Pods**: Cilium is now the first CNI to support TCP BBR congestion control for Pods in order to achieve significantly better throughput and lower latency for Pods exposed to lossy networks such as the Internet. (More details)
- **Bandwidth Manager promoted to Stable**: The bandwidth manager used to rate-limit Pod traffic and optimize network utilization has been promoted to stable (More details)
- **Dynamic Allocation of Pod CIDRs (beta)**: A new IPAM mode that improves Pod IP address space utilization due to dynamic assignment of Pod CIDRs to nodes. The latter can now allocate additional Pod CIDR pools dynamically to each node based on their usage. (More details)
- **Send ICMP unreachable host on Pod deletion**: When a Pod is deleted, Cilium can now install a route which informs other endpoints that the Pod is now unreachable. (More details)
- **AWS ENI prefix delegation**: Cilium now supports the AWS ENI prefix delegation feature, which effectively increases the allocatable IP capacity when running in ENI mode. (More details)
- **AWS EC2 instance tag filter**: A new Cilium Operator flag improves the scalability in large AWS subscriptions. (More details)

Tetragon:

- **Initial release**: Initial release of Tetragon that provides security observability and runtime enforcement using eBPF. (More details)

2.1.2 eBPF in Cilium

- $SRC_CILIUM/bpf

```
[mydev@fedora cilium-master]$ exa -T -L 1 bpf
bpf
├── bpf_alignchecker.c
├── bpf_features.h
├── bpf_host.c
├── bpf_lxc.c
├── bpf_network.c
├── bpf_overlay.c
├── bpf_sock.c
├── bpf_xdp.c
├── cilium-probe-kernel-hz.c
├── complexity-tests
├── COPYING
├── custom
├── ep_config.h
├── filter_config.h
├── include
├── init.sh
├── lib
├── LICENSE.BSD-2-Clause
├── LICENSE.GPL-2.0
├── Makefile
├── Makefile.bpf
├── netdev_config.h
├── node_config.h
├── sockops
├── source_names_to_ids.h
└── tests
[mydev@fedora cilium-master]$
```

```
[mydev@fedora cilium-master]$ tokei bpf
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 BASH                    1           14           10            3            1
 C                      22         8427         6042         1109         1276
 C Header              112        25950        16673         6421         2856
 Makefile                4          460          361           24           75
 ReStructuredText        1          131           97            0           34
 Shell                   1          574          448           53           73
 Plain Text             20           36            0           36            0
===============================================================================
 Total                 161        35592        23631         7646         4315
===============================================================================
[mydev@fedora cilium-master]$
```

- ...

Pure-Go eBPF Library

- **https://github.com/cilium/ebpf**

  eBPF is a pure Go library that provides utilities for loading, compiling, and debugging eBPF programs. It has minimal external dependencies and is intended to be used in long running processes.

  The library is maintained by Cloudflare and Cilium.

- **Packages**

  This library includes the following packages:

  - asm contains a basic assembler, allowing you to write eBPF assembly instructions directly within your Go code. (You don't need to use this if you prefer to write your eBPF program in C.)
  - cmd/bpf2go allows compiling and embedding eBPF programs written in C within Go code. As well as compiling the C code, it auto-generates Go code for loading and manipulating the eBPF program and map objects.
  - link allows attaching eBPF to various hooks
  - perf allows reading from a `PERF_EVENT_ARRAY`
  - ringbuf allows reading from a `BPF_MAP_TYPE_RINGBUF` map
  - features implements the equivalent of `bpftool feature probe` for discovering BPF-related kernel features using native Go.
  - rlimit provides a convenient API to lift the `RLIMIT_MEMLOCK` constraint on kernels before 5.11.

- **Requirements**

  - A version of Go that is supported by upstream
  - Linux >= 4.9. CI is run against kernel.org LTS releases. 4.4 should work but is not tested against.

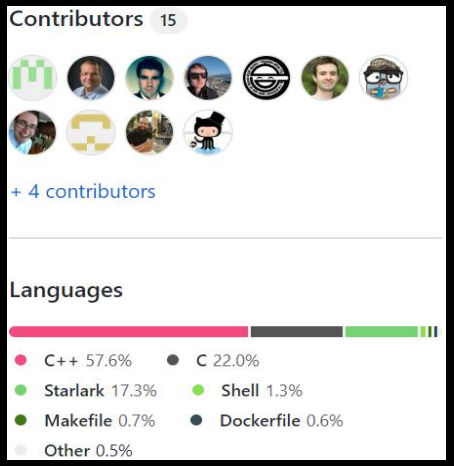
- **…**

proxy

- **https://github.com/cilium/proxy**

Envoy with Cilium filters

Envoy proxy for Cilium with minimal Envoy extensions and Cilium policy enforcement filters. Cilium uses this as its host proxy for enforcing HTTP and other L7 policies as specified in network policies for the cluster. Cilium proxy is distributed within the Cilium images.



pwru

- **https://github.com/cilium/pwru**

Packet, where are you? -- eBPF-based Linux kernel networking debugger.

`pwru` is an eBPF-based tool for tracing network packets in the Linux kernel with advanced filtering capabilities. It allows fine-grained introspection of kernel state to facilitate debugging network connectivity issues.

- **Requirements for running**

`pwru` requires >= 5.3 kernel to run. For `--output-skb` >= 5.9 kernel is required.

The following kernel configuration is required.

Option	Note
CONFIG_DEBUG_INFO_BTF=y	Available since >= 5.3
CONFIG_KPROBES=y	
CONFIG_PERF_EVENTS=y	
CONFIG_BPF=y	
CONFIG_BPF_SYSCALL=y	

You can use `zgrep $OPTION /proc/config.gz` to validate whether option is enabled.

- **Development dependencies**
  - Go >= 1.16
  - LLVM/clang >= 1.12
- **...**

2.1.3 Hubble

Overview

- https://github.com/cilium/hubble

Network, Service & Security Observability for Kubernetes using eBPF.

Hubble is a fully distributed networking and security observability platform for cloud native workloads. It is built on top of Cilium and eBPF to enable deep visibility into the communication and behavior of services as well as the networking infrastructure in a completely transparent manner.

Hubble can answer questions such as:

Service dependencies & communication map:

- What services are communicating with each other? How frequently? What does the service dependency graph look like?
- What HTTP calls are being made? What Kafka topics does a service consume from or produce to?

Operational monitoring & alerting:

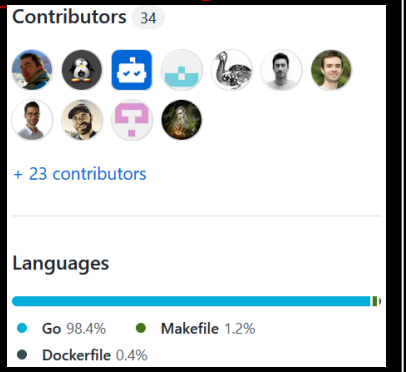
- Is any network communication failing? Why is communication failing? Is it DNS? Is it an application or network problem? Is the communication broken on layer 4 (TCP) or layer 7 (HTTP)?
- Which services have experienced a DNS resolution problems in the last 5 minutes? Which services have experienced an interrupted TCP connection recently or have seen connections timing out? What is the rate of unanswered TCP SYN requests?

Application monitoring:

- What is the rate of 5xx or 4xx HTTP response codes for a particular service or across all clusters?
- What is the 95th and 99th percentile latency between HTTP requests and responses in my cluster? Which services are performing the worst? What is the latency between two services?

Security observability:

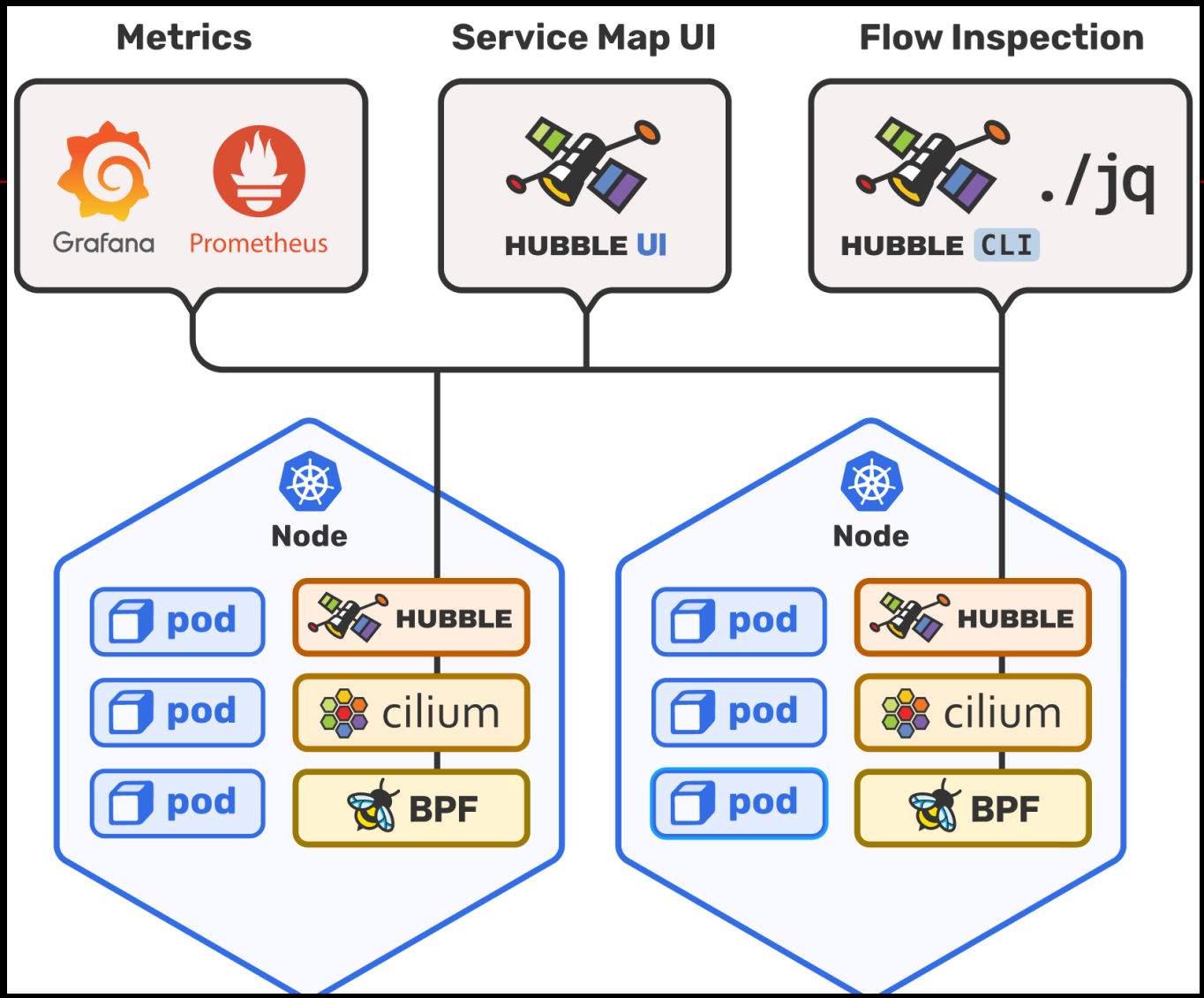
- Which services had connections blocked due to network policy? What services have been accessed from outside the cluster? Which services have resolved a particular DNS name?



- https://github.com/cilium/hubble-ui

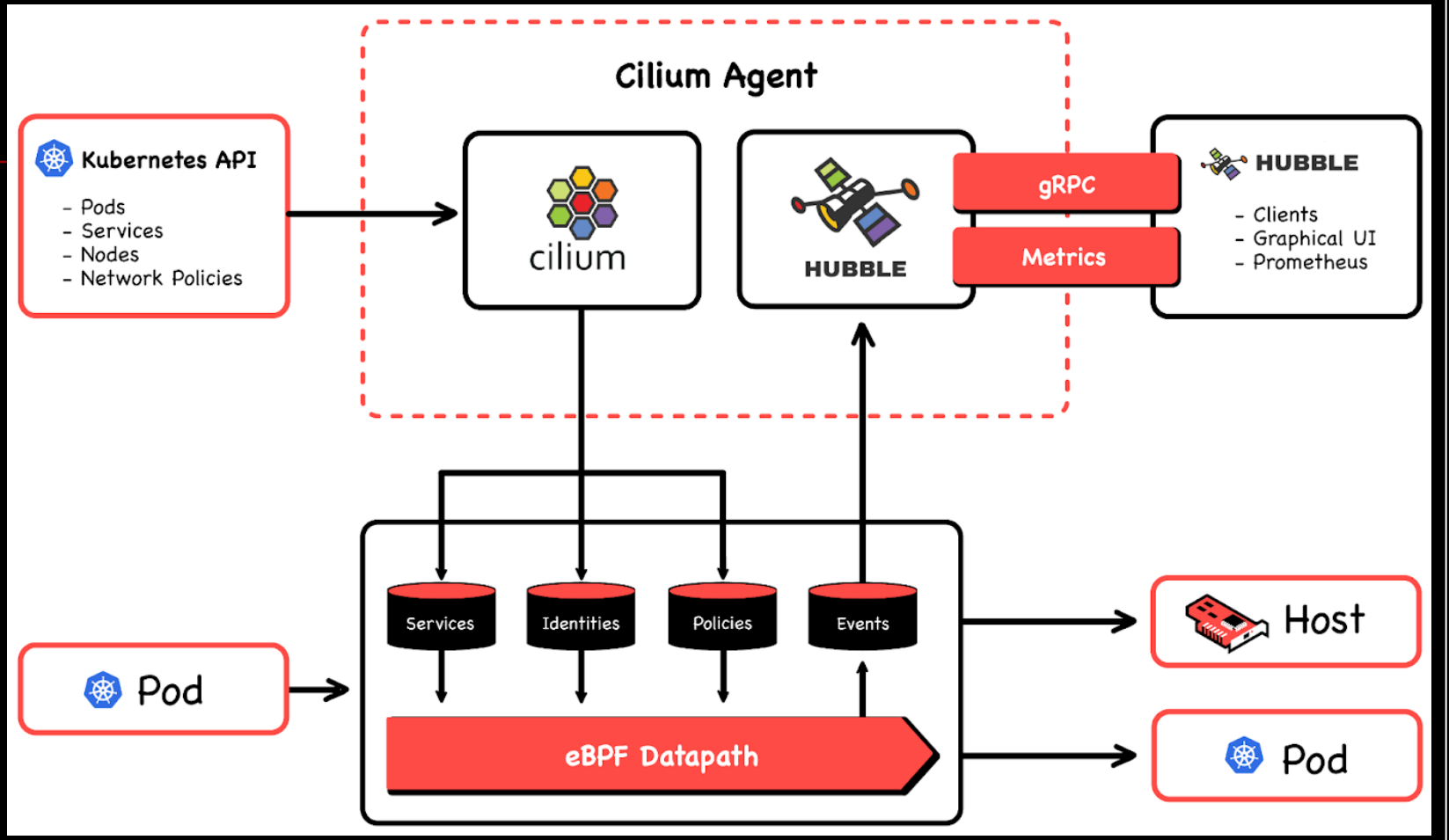
- https://docs.cilium.io/en/stable/gettingstarted/hubble/#deploy-cilium-and-hubble
- https://www.datadoghq.com/blog/cilium-metrics-and-architecture
- https://www.datadoghq.com/blog/monitor-cilium-and-kubernetes-performance-with-hubble/

Architecture & Design



Source: https://github.com/cilium/hubble/blob/master/Documentation/images/hubble_arch.png

How it works



Source: https://blog.container-solutions.com/ebpf-cloud-native-tools-an-overview-of-falco-inspektor-gadget-hubble-and-cilium

2.1.4 Tetragon

Overview

- **https://github.com/cilium/tetragon**

eBPF-based Security Observability and Runtime Enforcement.

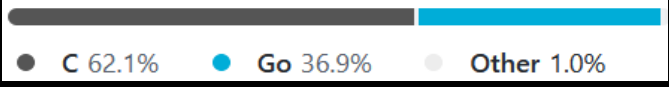
Cilium's new Tetragon component enables powerful realtime, eBPF-based Security Observability and Runtime Enforcement.

Tetragon detects and is able to react to security-significant events, such as

- Process execution events
- System call activity
- I/O activity including network & file access

When used in a Kubernetes environment, Tetragon is Kubernetes-aware - that is, it understands Kubernetes identities such as namespaces, pods and so-on - so that security event detection can be configured in relation to individual workloads.

Languages

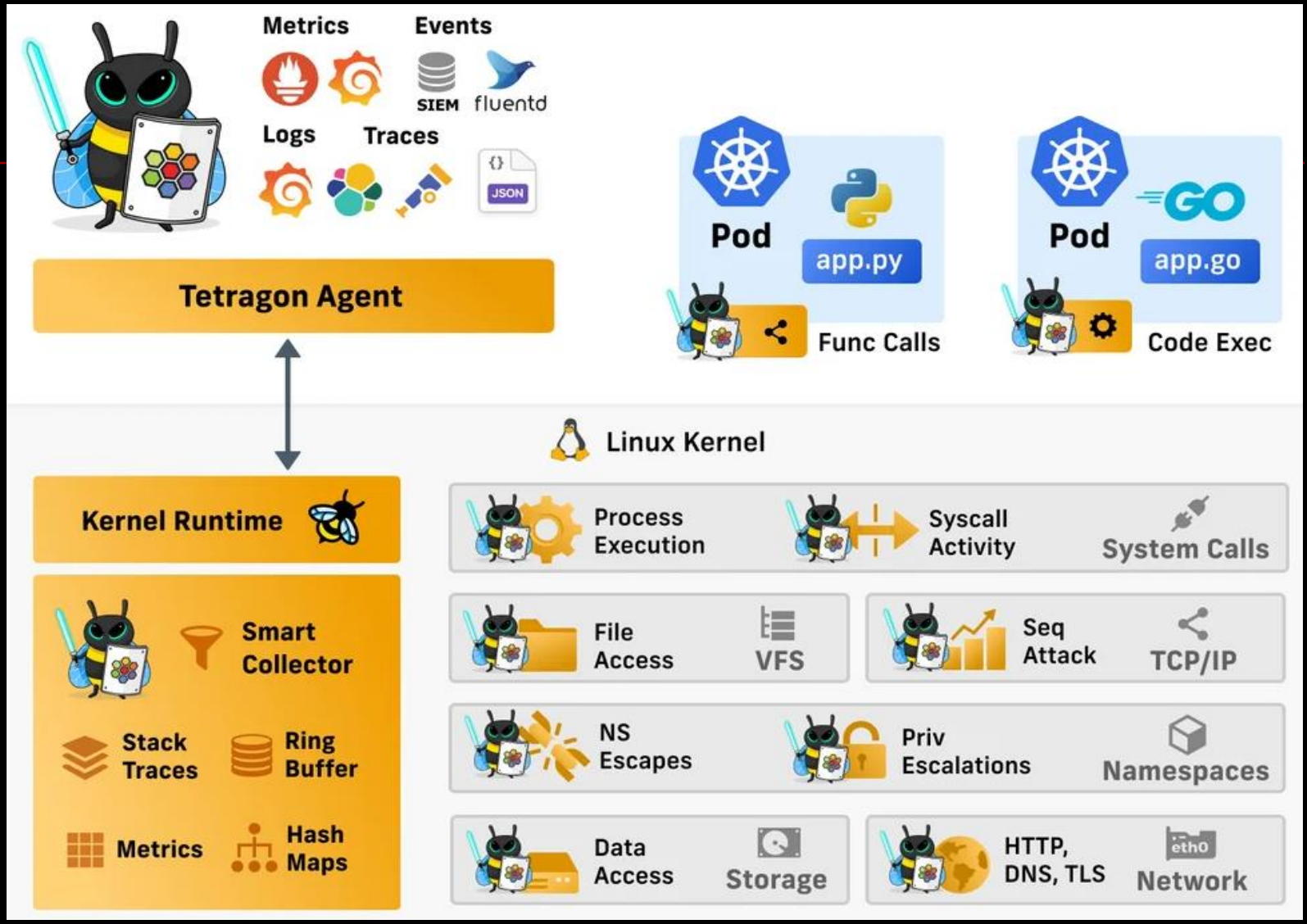


Requirements

The base kernel should support BTF or the BTF file should be placed where Tetragon can read it.

- **https://github.com/cilium/tetragon/blob/main/docs/contributing/development/README.md**

Architecture & Design



Source: https://github.com/cilium/tetragon/blob/main/docs/images/smart_observability.png

2.1.5 eBPF-powered Cilium Service Mesh

Overview

- **https://github.com/cilium/cilium-service-mesh-beta (ended)**

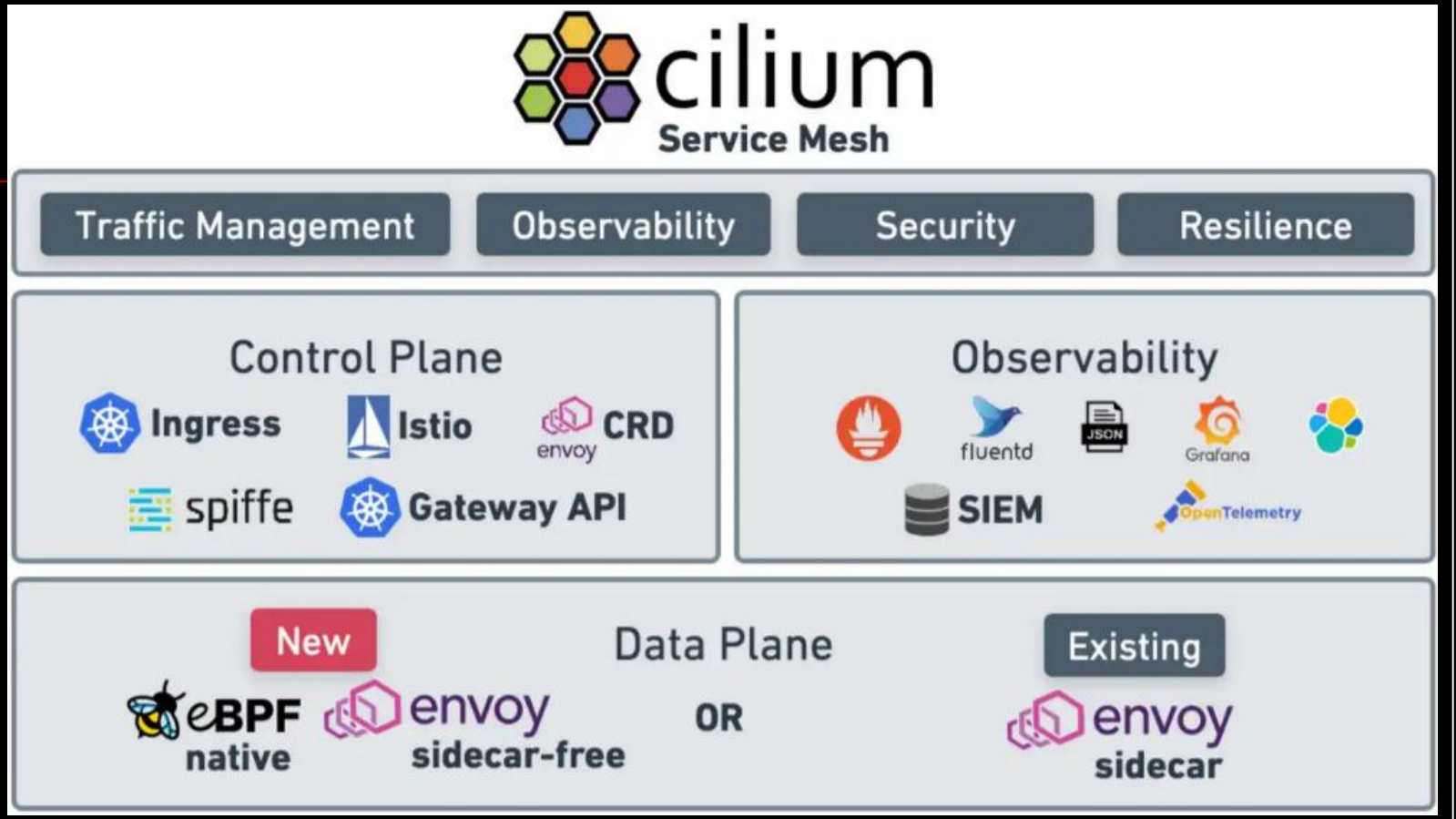
> The beta phase of Cilium Service Mesh has ended. Cilium service mesh is now available in the regular Cilium release (1.12)

- **https://docs.cilium.io/en/latest/network/servicemesh/**
-

> We are excited about this initial release of Cilium Service Mesh on top of the existing networking, security, and observability function of Cilium. It gives users choice:
>
> - **Control Plane:** Choice of control plane options for the ideal balance of complexity and richness. From simpler options such as Ingress and Gateway API, to richer options with Istio, to the full power of Envoy via the Envoy CRD.
> - **Sidecar vs Sidecar-free:** Choice of a datapath with or without sidecars. Sidecars with VM-style resource isolation at increased overhead and cost, or container-style shared resources at the cost of requiring to manage the shared resource usage.

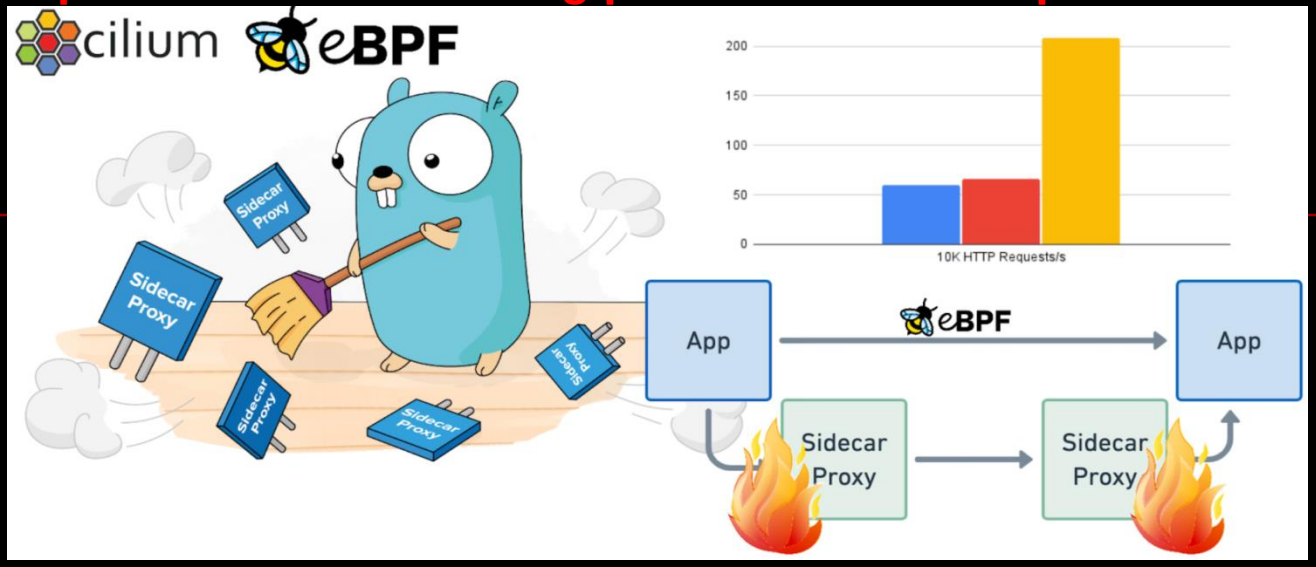
Source: https://isovalent.com/blog/post/cilium-service-mesh/

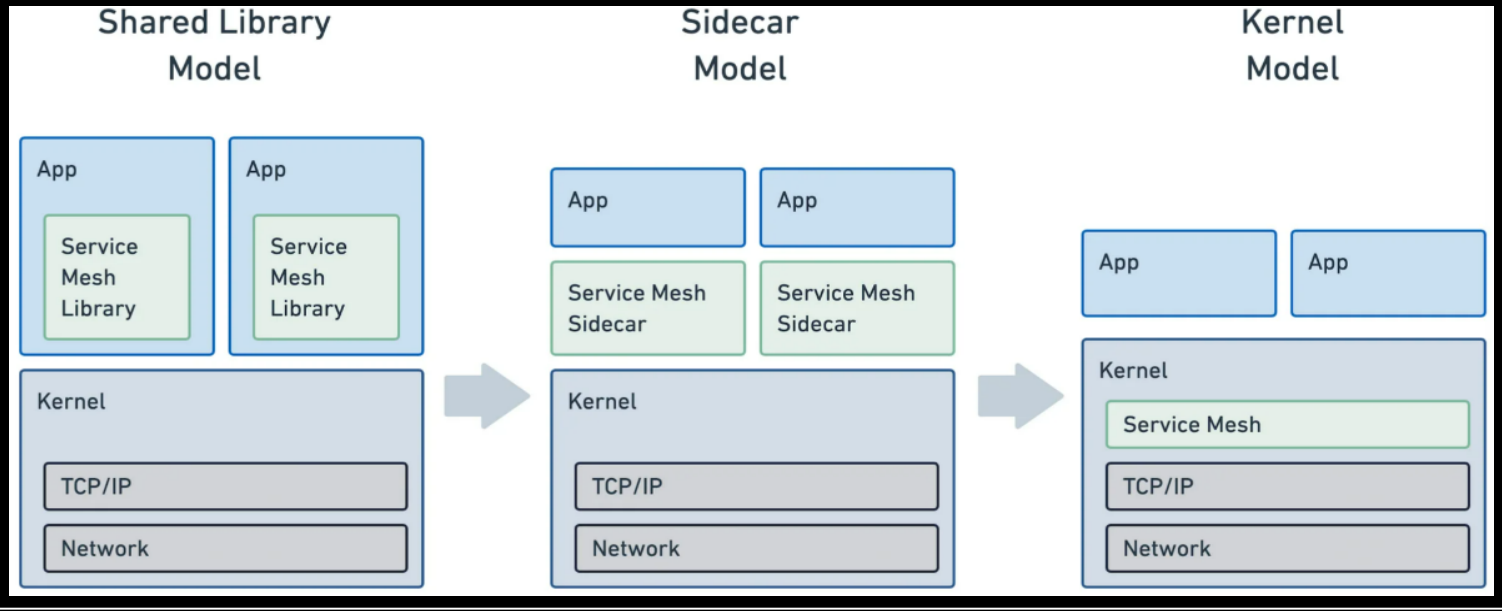
Architecture & Design



Source: https://isovalent.com/blog/post/cilium-service-mesh/

- https://isovalent.com/blog/post/2021-12-08-ebpf-servicemesh/

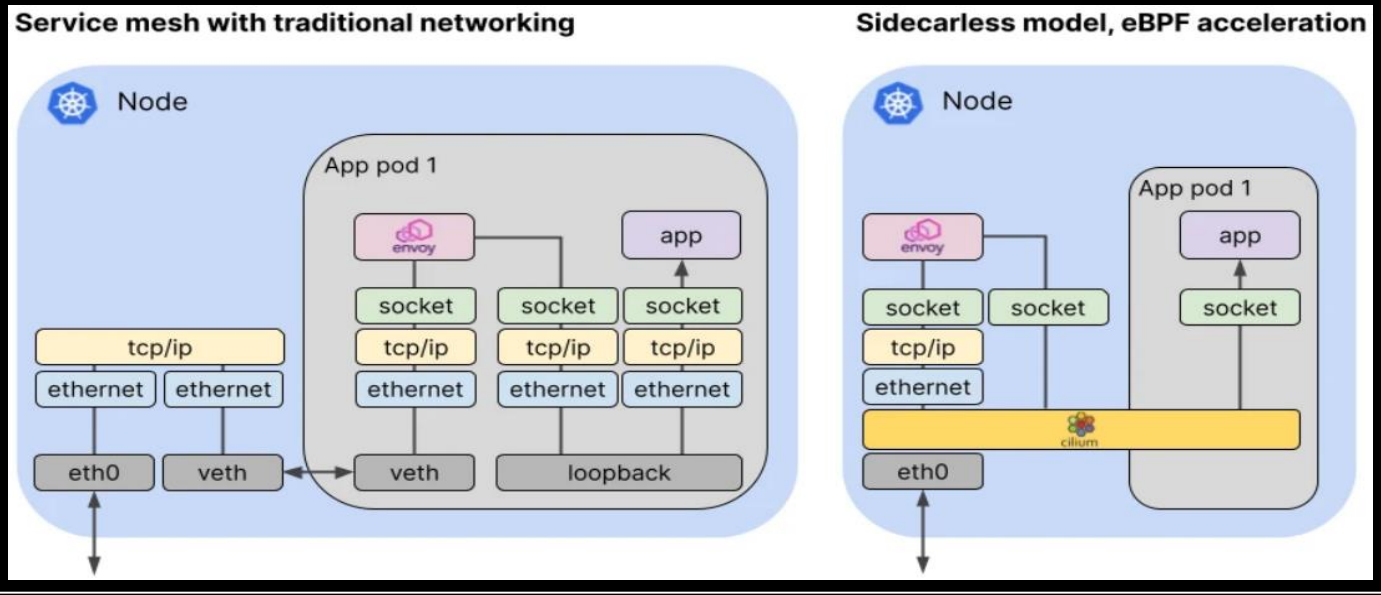




- **https://thenewstack.io/how-ebpf-streamlines-the-service-mesh/**

The term “Service Mesh” encompasses a wide range of features, including service discovery, encryption, service authentication, load balancing, observability, canary roll-outs and more. Some of these features overlap with established Cilium capabilities - for example, Cilium has offered load balancing, Kubernetes service awareness, multi-cluster connectivity, and visibility of network traffic at layer 3-7, for ages. Cilium already uses Envoy for L7 policy and observability for some protocols, and this same component is used as the sidecar proxy in many popular Service Mesh implementations. So it's a natural step to extend Cilium to offer more of the features commonly associated with Service Mesh.

In a typical Service Mesh, all network packets need to pass through a sidecar proxy container on their path to or from the application container in a Pod. In Cilium Service Mesh, we’re moving that proxy container onto the host and kernel so that sidecars for each application pod are no longer required. Because eBPF allows us to intercept packets at the socket as well as at the network interface, Cilium can dramatically shorten the overall path for each packet. (Read more about sidecarless, eBPF-based Service Mesh.)



3) eBPF for Edge Computing

Overview

- **https://en.wikipedia.org/wiki/Edge_computing**

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the sources of data. This is expected to improve response times and save bandwidth.[1] It is an architecture rather than a specific technology.[2] It is a topology- and location-sensitive form of distributed computing.

The origins of edge computing lie in content distributed networks that were created in the late 1990s to serve web and video content from edge servers that were deployed close to users.[3] In the early 2000s, these networks evolved to host applications and application components at the edge servers,[4] resulting in the first commercial edge computing services[5] that hosted applications such as dealer locators, shopping carts, real-time data aggregators, and ad insertion engines.[4]

Internet of things (IoT) is an example of edge computing. A common misconception is that edge and IoT are synonymous.[6]

- **https://en.wikipedia.org/wiki/Multi-access_edge_computing**

Multi-access edge computing (**MEC**), formerly **mobile edge computing**, is an ETSI-defined[1] network architecture concept that enables cloud computing capabilities and an IT service environment at the edge of the cellular network[2][3] and, more in general at the edge of any network. The basic idea behind MEC is that by running applications and performing related processing tasks closer to the cellular customer, network congestion is reduced and applications perform better. MEC technology is designed to be implemented at the cellular base stations or other edge nodes, and enables flexible and rapid deployment of new applications and services for customers. Combining elements of information technology and telecommunications networking, MEC also allows cellular operators to open their radio access network (RAN) to authorized third parties, such as application developers and content providers.

Technical standards for MEC are being developed by the European Telecommunications Standards Institute, which has produced a technical white paper about the concept.[4]

- …

3.2 From our perspective

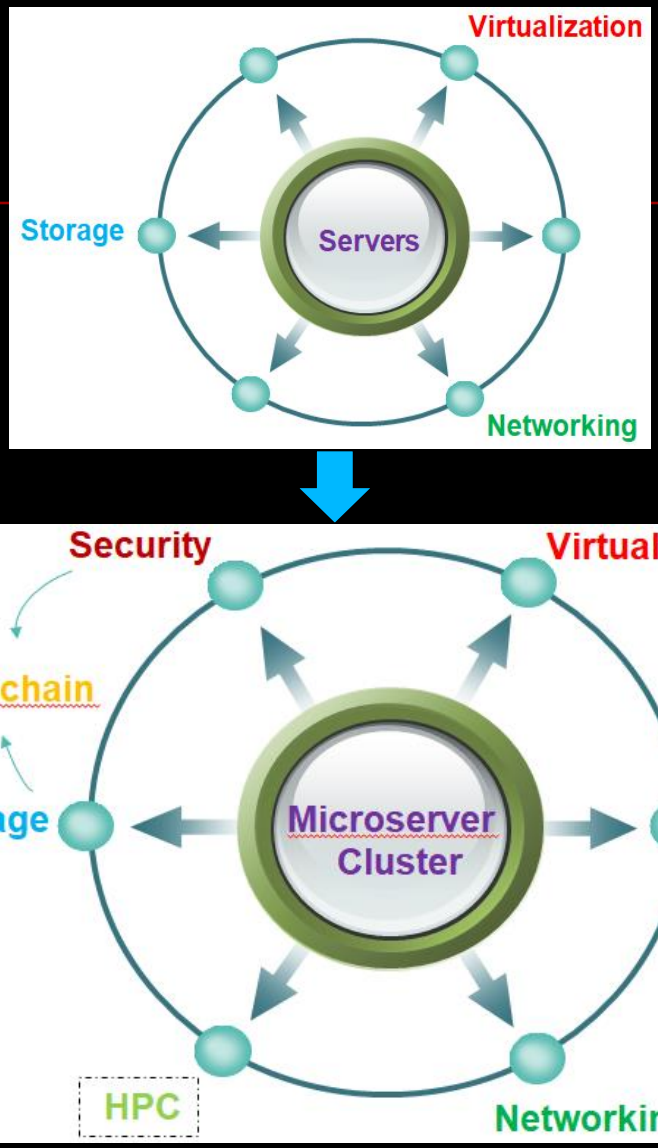
3.2.1 Previous consideration

OpenInfra Days China
Shanghai 2019

Rethinking Hyper-Converged Infrastructure for Edge Computing

Feng Li (李枫)
hkli2013@126.com
Nov 5，2019

- Trend(from my point of view)



3.2.2 Original Design

- An eBPF-centric new lightweight for Edge Computing that are devided into four stages

Key points

4) Summary

- Global Edge Computing Market is keep on growing, which mainly driven by the burst of Internet of Thing
- Hardware and software vendors in Edge computing like "Let a hundred flowers bloom"
- Currently, there is no dominant solution provider at Edge like what AWS, GCP, and Aliyun does at Cloud
- Due to the diverse requirement for Edge Computing, and considering it may meet the resource limitation problem when comparing with Cloud Computing, so a lightweight and flexible solution with high cost–performance ratio is still very attractive

2) Hardware Platform

- ARM is the best choice in current stage (from my point of view)
  - high cost–performance ratio development boards
  - an increasingly mature ecosystem
    - a lot of vendors to choose from
    - lower power consumption
  - ARM is ruling IoT and Embedded
  - the major FPGA vendor Xilinx uses ARM as hardware cores on their Reconfigurable Computing platform
  - …
- may migrate to ARM, X86, RISC-V Hybrid Architecture in the near future, but ARM is still first class

Source: "Rethinking Hyper-Converged Infrastructure for Edge Computing", Feng Li, OpenInfra Days China 2019.

II. Overall Design

1) Design Goals & Principles

Goals

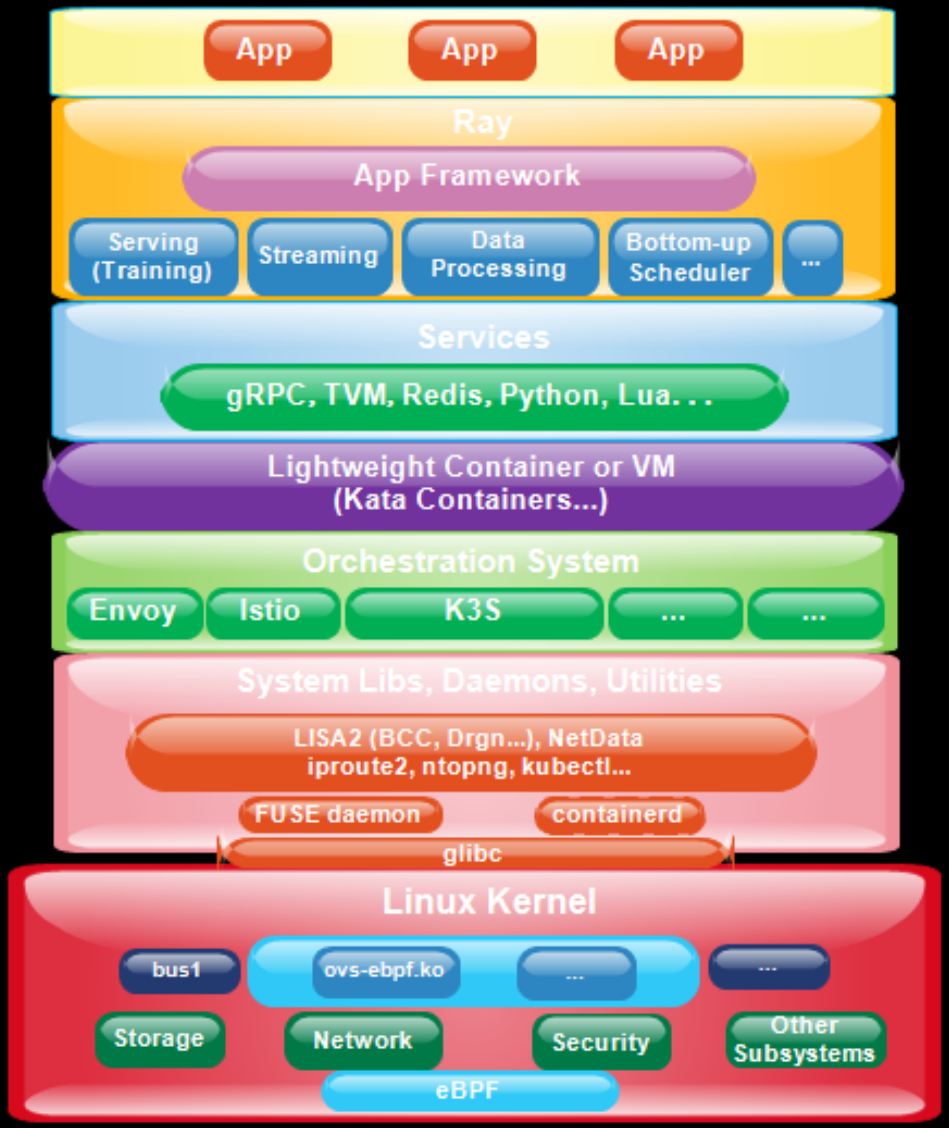
- a lightweight Edge Computing solution with high cost–performance ratio
- flexible and modular architecture
- scale out, not scale up
- meet the trend for Hyper-Converged Infrastructure at Edge
- …

Principles

- no JVM based project is considered
  (so Spark, Flink, Kafka, ElasticSearch are excluded…)
- reduce the dependencies for Go-based project to least
  (though it is difficult to do so…)
- make project code reusable and self-contained as much as possible
- …

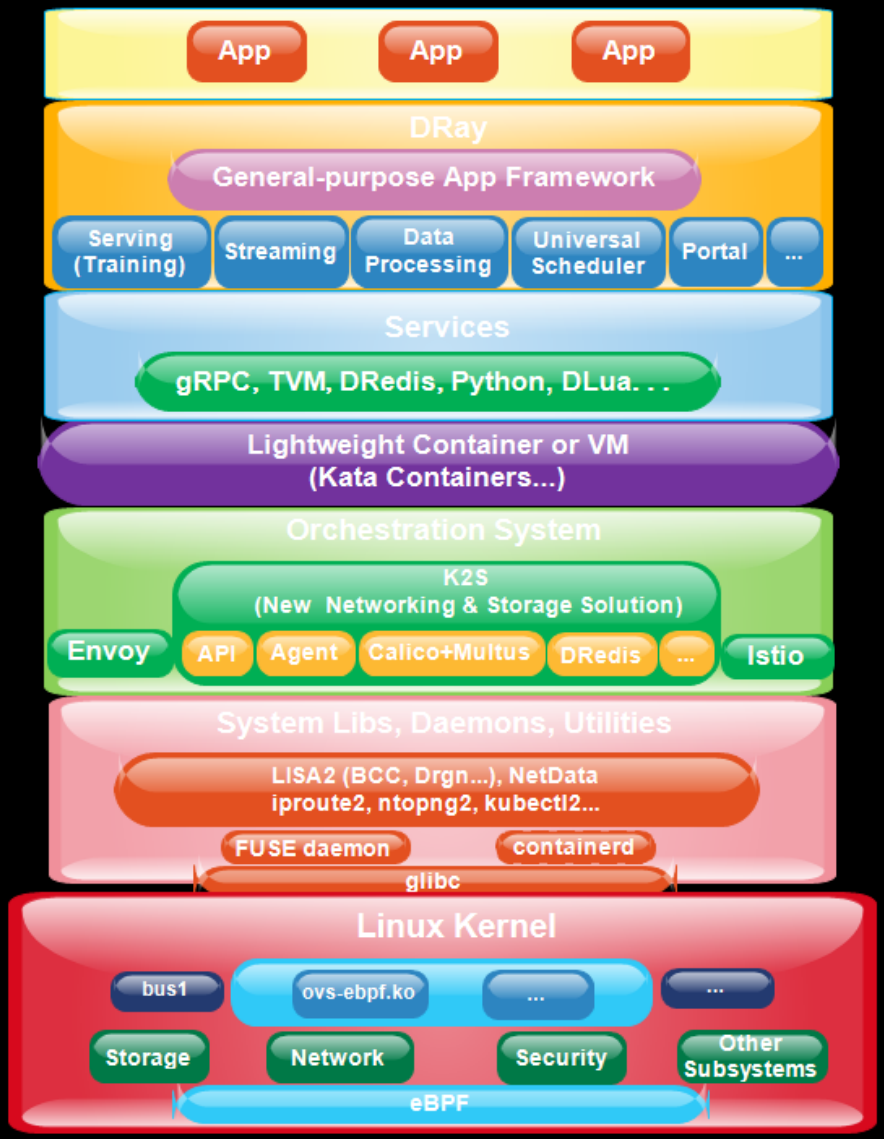
Source: "Rethinking Hyper-Converged Infrastructure for Edge Computing", Feng Li, OpenInfra Days China 2019.

Stage I



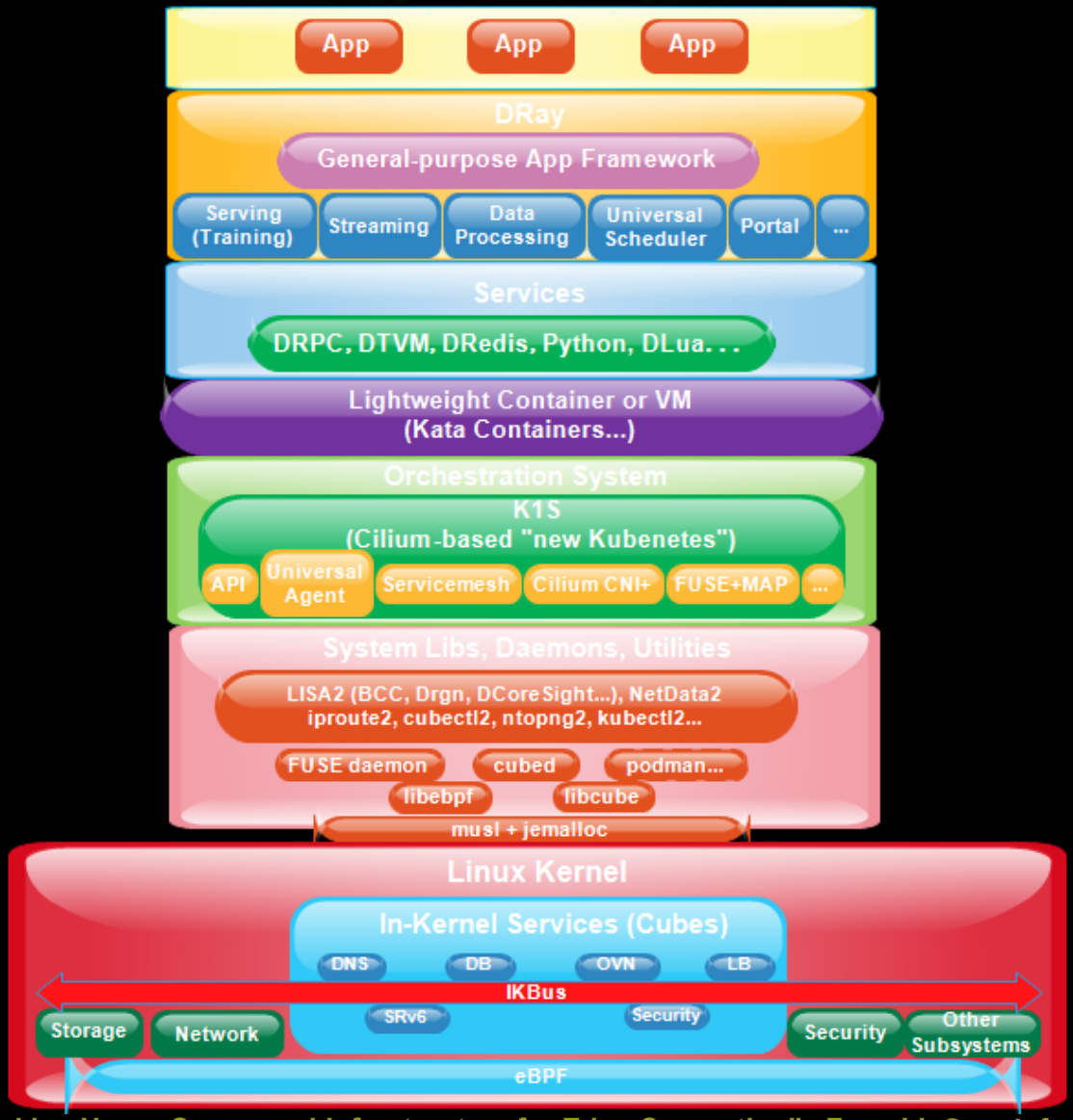
Source: "Rethinking Hyper-Converged Infrastructure for Edge Computing", Feng Li, OpenInfra Days China 2019.

Stage II



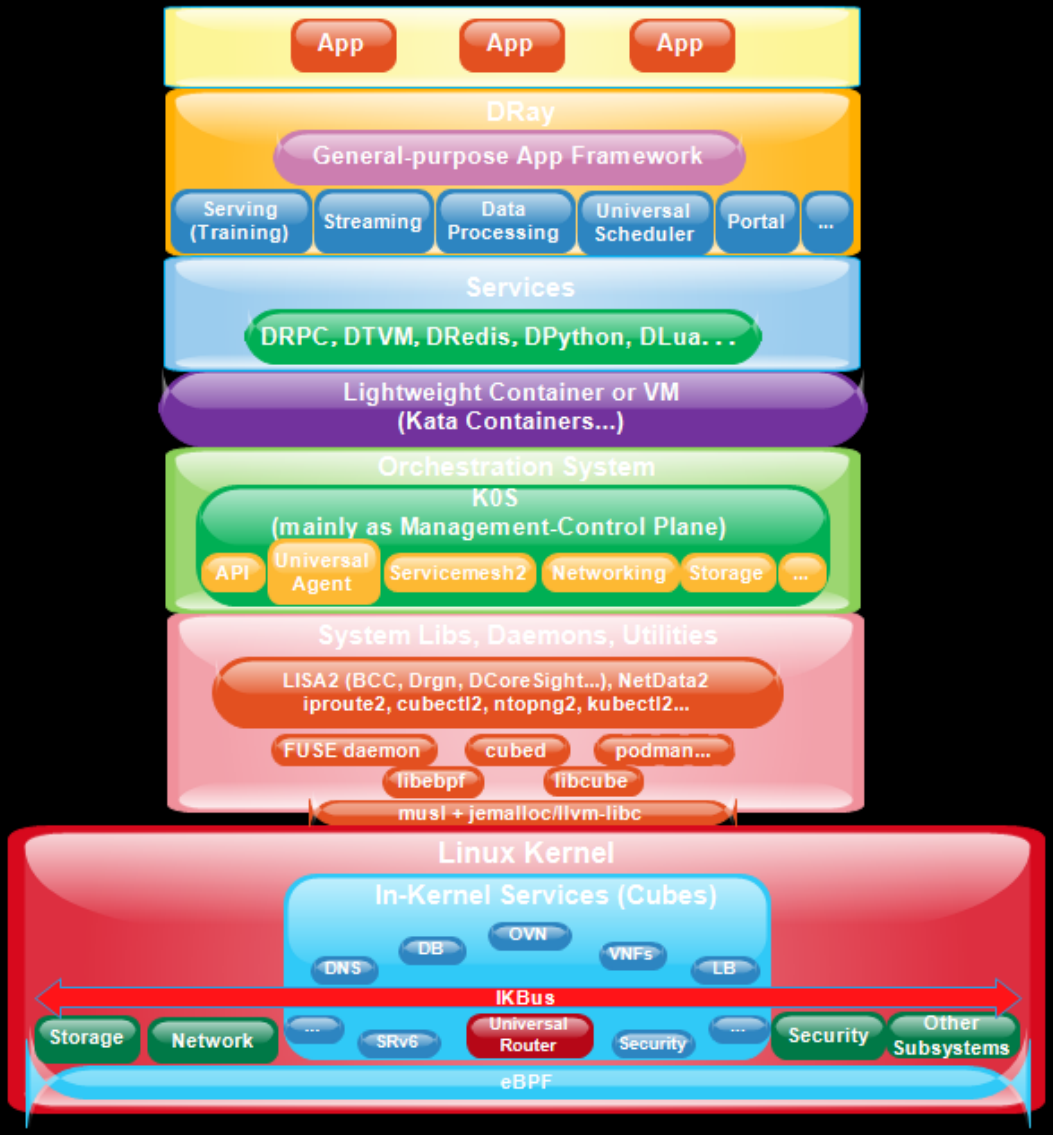
Source: "Rethinking Hyper-Converged Infrastructure for Edge Computing", Feng Li, OpenInfra Days China 2019.

Stage III



Source: "Rethinking Hyper-Converged Infrastructure for Edge Computing", Feng Li, OpenInfra Days China 2019.

Stage 4



Source: "Rethinking Hyper-Converged Infrastructure for Edge Computing", Feng Li, OpenInfra Days China 2019.

3.2.3 Current scheme

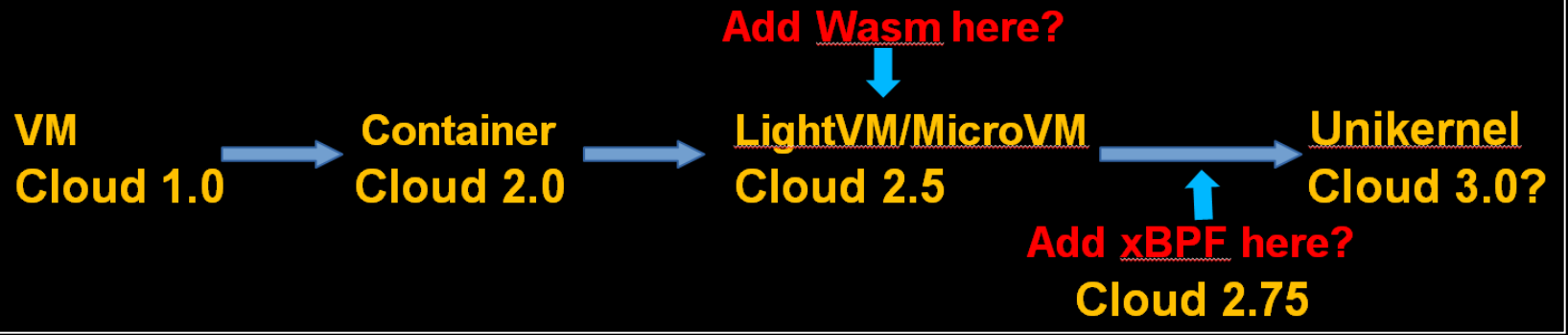
- Continuously updated "Revisiting the eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing" :
  e.g., https://github.com/XianBeiTuoBaFeng2015/MySlides/blob/master/Conf/2021/K%2BSummit2021__Revisiting%20eBPF-centric%20New%20Design%20for%20HCI%20and%20Edge%20Computing__FengLi-updated20220405p.pdf
- The third-round discussion on this topic will be divided into two series "ARM + Python + Rust + Lua + GraalVM + …" and "RISC-V + Python +  + Lua +  + …" according to different technology roadmap.
- Please look forward to our upcoming follow-ups…

V. Wrap-up

- User Space & Kernel Space repartition in Linux has a long history, now eBPF is driving it.

- Nearly every subsystem of Linux is or will be effected by eBPF.
- Changing the way you think about Linux Kernel development.
  E.g., Kernel Space & User Space Instrumentation:

- Cloud era: Past, Present, and Future(from our perspective):



- **The 2022 Linux Kernel Maintainers Summit**

  https://events.linuxfoundation.org/linux-kernel-maintainer-summit/

- **Our follow-ups for eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing will be divided into "HW/SW Tech Stack 101", "eBPF 101", "eBPF-based Projects", "eBPF: Past, Present, and Future" and more, then itself will only focus on design and implementation of the two schemes according to different technology roadmap as described previously.**

Q & A

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://en.wikipedia.org/wiki/GNU_Compiler_Collection
- https://en.wikipedia.org/wiki/LLVM
- https://en.wikipedia.org/wiki/Kernel_debugger
- https://www.quora.com/How-do-I-view-the-kernel-source-code-in-Linux
- https://pentera.io/blog/the-good-bad-and-compromisable-aspects-of-linux-ebpf/
- https://www.infoq.com/articles/gentle-linux-ebpf-introduction/
- https://cloudyuga.guru/hands_on_lab/ebpf-intro
- https://oswalt.dev/2021/01/introduction-to-ebpf/
- https://tinyurl.com/ebpf-linux
- https://github.com/iovisor/bpftrace/blob/master/docs/reference_guide.md
- https://cloudblogs.microsoft.com/opensource/2022/02/22/getting-linux-based-ebpf-programs-to-run-with-ebpf-for-windows/
- https://fly.io/blog/bpf-xdp-packet-filters-and-udp/
- https://www.ferrisellis.com/content/ebpf_past_present_future/
- https://devopsdays.org/events/2022-portugal/program/rinor-maloku

- https://www.ebpf.top/
- …