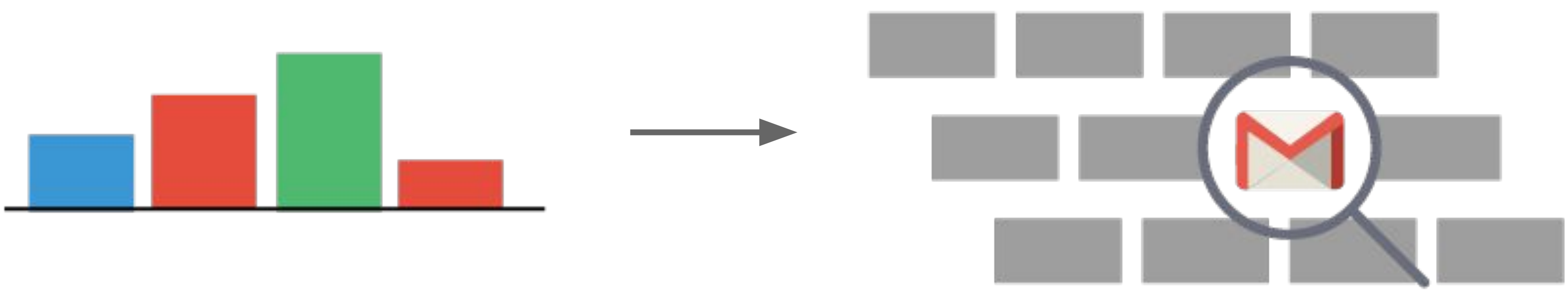
Using eBPF for network traffic analysis

Samuele Sabella
Luca Deri

Network Monitoring

In the past few years the need to better understand what's happening in our networks has lead from tools for volumetric data analysis to what is now called "deep packet inspection"



Containerization
era

Containers are dynamical entities that provide microservices and can be contracted or expanded (i.e. elastic computing) on needs (e.g. kubernetes).

Monitoring the system from the outside (i.e. looking at network packets), is no longer enough.

Monitoring
Containers

The “photography” we take by looking at traffic can be out of date.

The information we gather looking at the packets only, are not complete:

- How much traffic belongs to user X?
- Calculating latency by looking at packets is reconstructing what we think is happening at a lower level

It is not an absolute view, we don’t know what is happening in what isn’t our domain!

Not a new idea:
sysdig

Provides a way to observe the system at the kernel system call level.

We receive system call events, that however are difficult and time consuming to interpret:

- We work at a high level
- Runtime complexity, heavy load on CPU

It is a very good tool but with some limitations.

eBPF: what is and how can be used?

In 1997, it was introduced in Linux kernel as a technology for in-kernel packet filtering. The authors are Steven McCanne and Van Jacobson from Lawrence Berkeley Laboratory.

eBPF extended what bpf virtual machine could do, allowing it to run other kind of events and take several action other than filtering

eBPF: lot of very useful tools

- tcplife: Trace the lifespan of TCP sessions and summarize.
- tcptop: Summarize TCP send/recv throughput by host.
- biolatency: Summarize block device I/O latency as a histogram.
- filetop: file reads and writes by process.

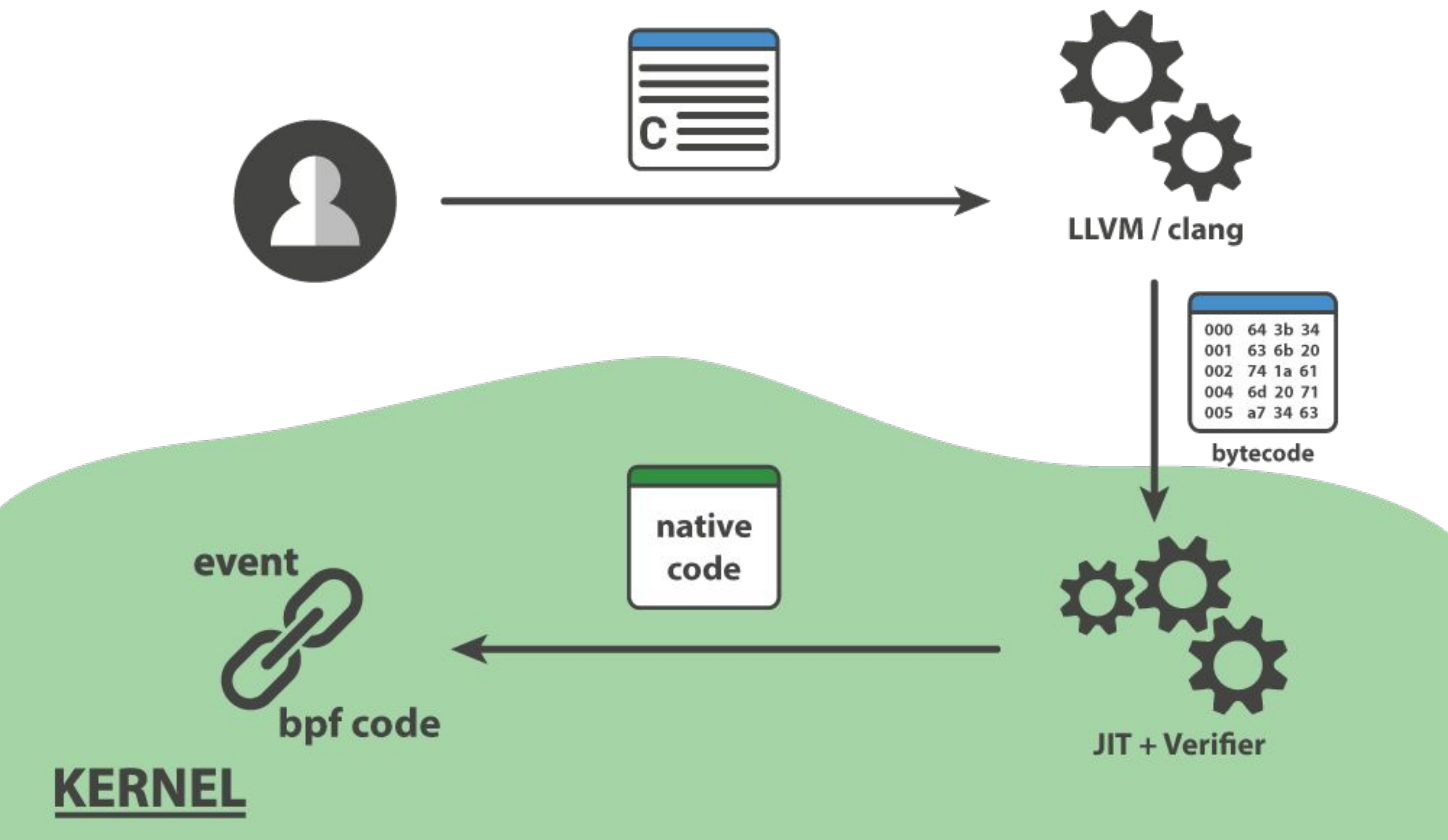
A toolkit for ebpf:
bcc

An easy to use toolkit to write eBPF programs that offers a front-end interface in different languages:

- Python
- Lua
- C++
- Rust

The repository offers a lot of examples on various topics

BPF: how it works



eBPF/bcc: basic usage

Events

- Kprobes
- Kretprobes
- Uprobes
- Uretprobs

Output

- printk
- perf_output

Maps

- Hash tables
- Histograms
- Lru hash

Helper functions

- bpf_get_current_task
- bpf_ktime_get_ns
- bpf_get_current_comm

eBPF: limitations

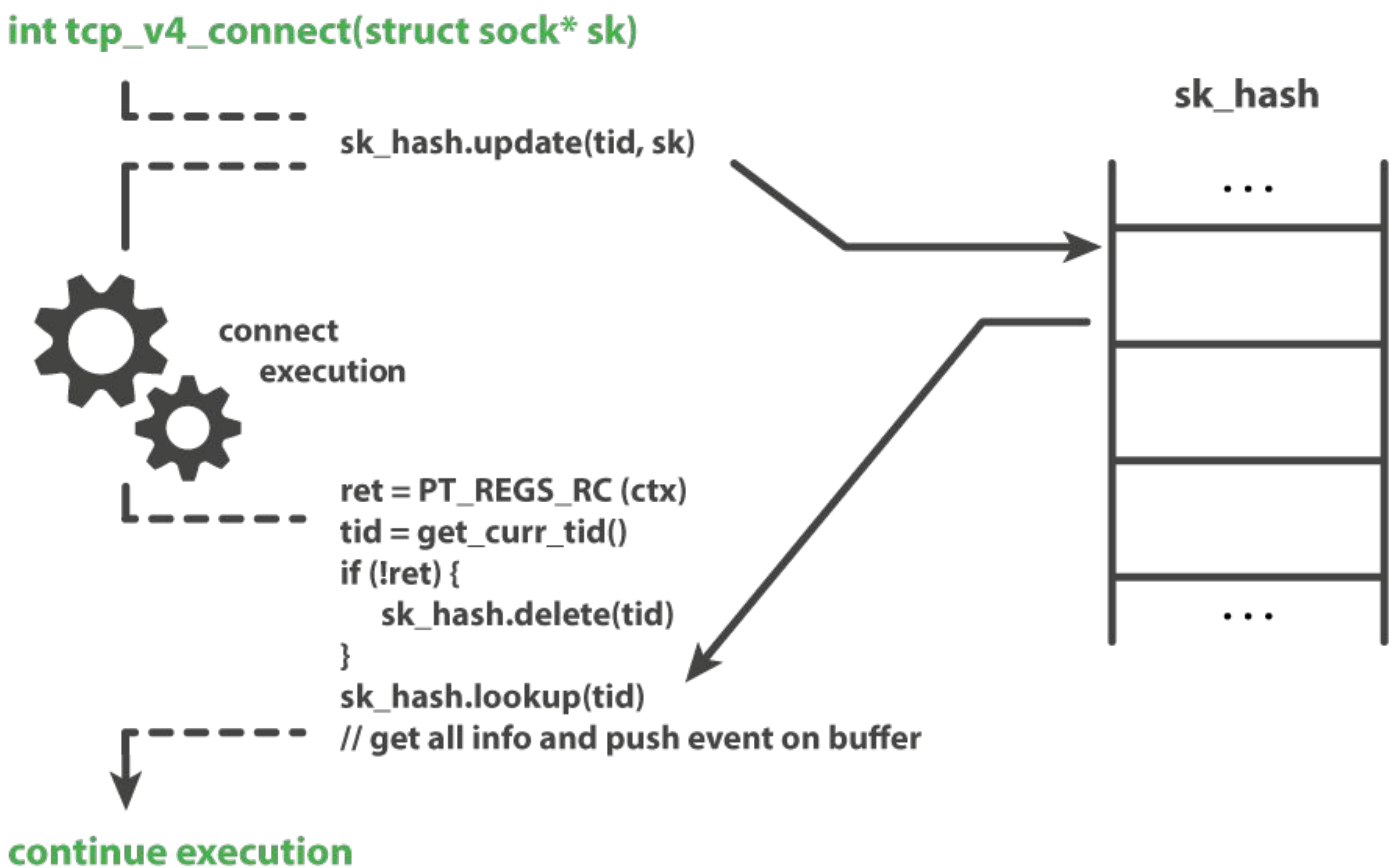
- eBPF and bcc are not mature projects
- Difficult to use
- Kernel functions available to use are the one determined by the flag prog_type
- We can’t to do cycles
- VM is read only in kprobes!
- To access data external to ebpf stack we must use bpf_probe_read (not always necessary, the compiler may provide us support)

ebpflow: our objectives

Create an in-kernel flow monitoring tool for traffic analysis which can observe the system and take actions within the kernel.

- Reliable and trustworthy information on the status of the system when events take place.
- Low overhead event-based monitoring
- Information on users, network statistics, containers and processes

ebpflow: basic tcp



ebpflow: how to spot containers?

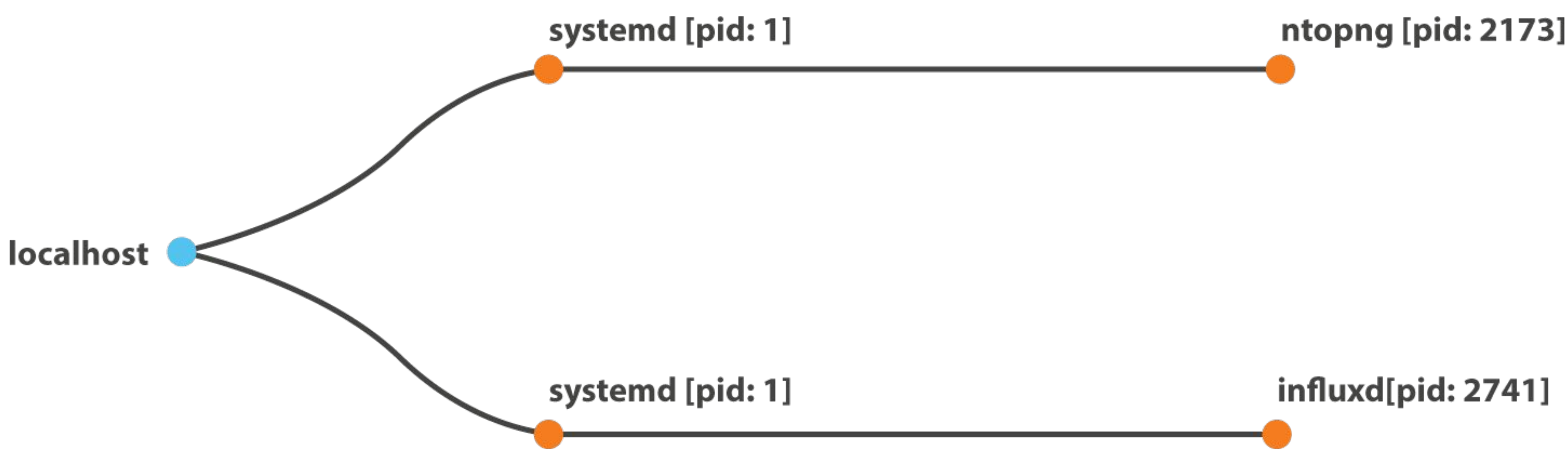
Container can be spotted by looking at proc/cgroup, however retrieving information from there is a too slow operation

Containers are processes isolated with the use of namespaces and cgroups

- We can navigate through kernel data structures and read the namespaces
- We extract information at kernel level

ebpflow: ntop integration

- Low overhead
- Faithful picture of the state of the system
- Per user flow informations
- We can build the process hierarchy for each flow



ntop

Flows | Hosts | Interfaces

Active Flows

10 Hosts

	Application	L4 Proto	Client	Server	Duration	Breakdown
Info	Redis	TCP	localhost :42644	localhost ::6379	00:22	Client Server
Info	DNS	UDP	localhost :51259	localhost::domain	< 1 sec	Client Server
Info	DNS.ntop	UDP	localhost :59163	localhost::domain	< 1 sec	Client Server
Info	HTTP	TCP	localhost :39174 [root >_ ntopng]	localhost ::8086 [influxdb >_ influxd]	< 1 sec	Client Ser
Info	HTTP	TCP	localhost :39172 [root >_ ntopng]	localhost ::8086 [influxdb >_ influxd]	< 1 sec	Client Ser
Info	HTTP	TCP	localhost :39170 [root >_ ntopng]	localhost ::8086 [influxdb >_ influxd]	< 1 sec	Client Ser
Info	DNS	UDP	localhost :40920	localhost::domain	< 1 sec	Client Server
Info	Redis	TCP	localhost :42646	localhost ::6379	< 1 sec	Client Serve

Showing 1 to 8 of 8 rows. Idle flows not listed.

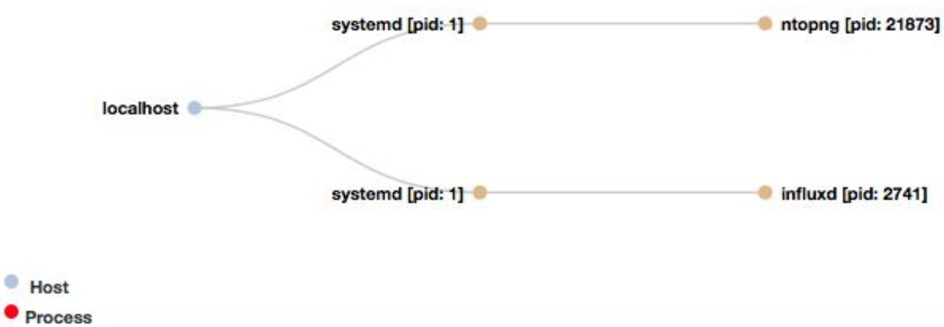
ntopng Enterprise Edition v.3.7.181015
User admin Interface lo

87.35 kbit/s [66 pps]

0 bps
0 bps

21:
2

TCP Flags	Client → Server: FIN SYN PUSH ACK	Client ← Server: FIN SYN PUSH
	This flow is completed and will expire soon.	
Flow Status	Normal	



Client Process Information		
User Name	root	
Process PID/Name	21873/ntopng [son of 1/systemd]	
Server Process Information		
User Name	influxdb	
Process PID/Name	2741/influxd [son of 1/systemd]	
HTTP	**HTTP Method**	GET
	Server Name	localhost
	URL	localhost/query?db=ntopng&q=S
	Response Code	200
Dump Flow Traffic		

Demo

Future work

- In Linux Kernel version 4.16 a new functionality has been added: bpf_override_return
  - Provides a way to override the return value of functions
  - The kernel function has to be whitelisted to allow error injection with: ALLOW_ERROR_INJECTION
  - It is supported only by few function (e.g. open_ctree) but in the future we hope also the networking functions will be supported

Conclusions

eBPF and bcc are great and powerful tools. However, due to the fact that they are not mature project, they are not stable and lack of some basic features.

Some workaround are often needed.

They can offer a different point of view from the one provided by looking only at traffic that goes through the system

References

BCC github repository:

https://github.com/iovisor/bcc

Brendan Gregg blog

http://www.brendangregg.com/

Reading material

https://qmonnet.github.io/whirl-offload/2016/09/01/dive-into-bpf/

References

eBPF intro:

https://www.netronome.com/blog/bpf-ebpf-xdp-and-bpfilter-what-are-these-things-and-what-do-they-mean-enterprise/

Cool blog by Julia Evans:

https://jvns.ca/blog/2017/07/05/linux-tracing-systems