Revalidator Tracepoint Implementation in Open vSwitch

Kevin Sprague

What is eBPF?

- (Over)simply: Event-based way for users to run custom code in kernel or applications
- 4 (main) types of attach points, along two axes

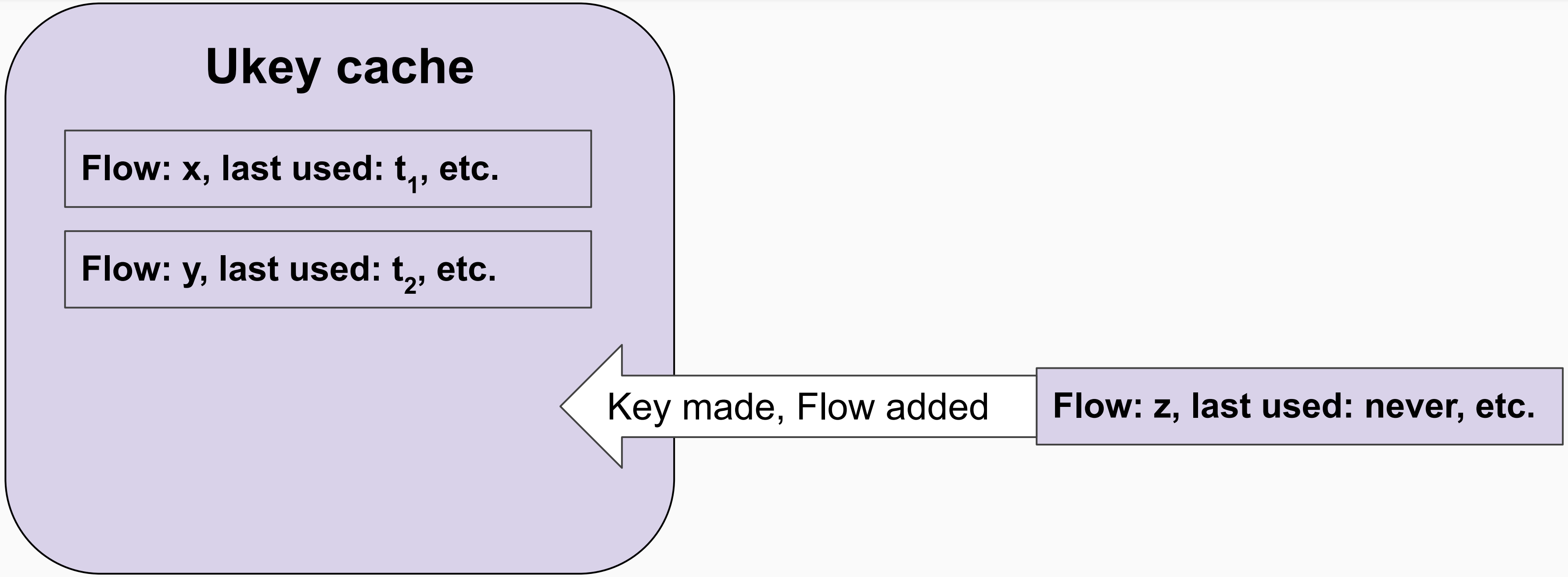
	Kernel	Userspace
Dynamic	Kprobe	Uprobe
Static	Tracepoint	USDT

- Everything bpf-related is privilege-enforced by default
- Verifier ensures that eBPF program is “safe” [1]
- Multiple development frameworks [2]

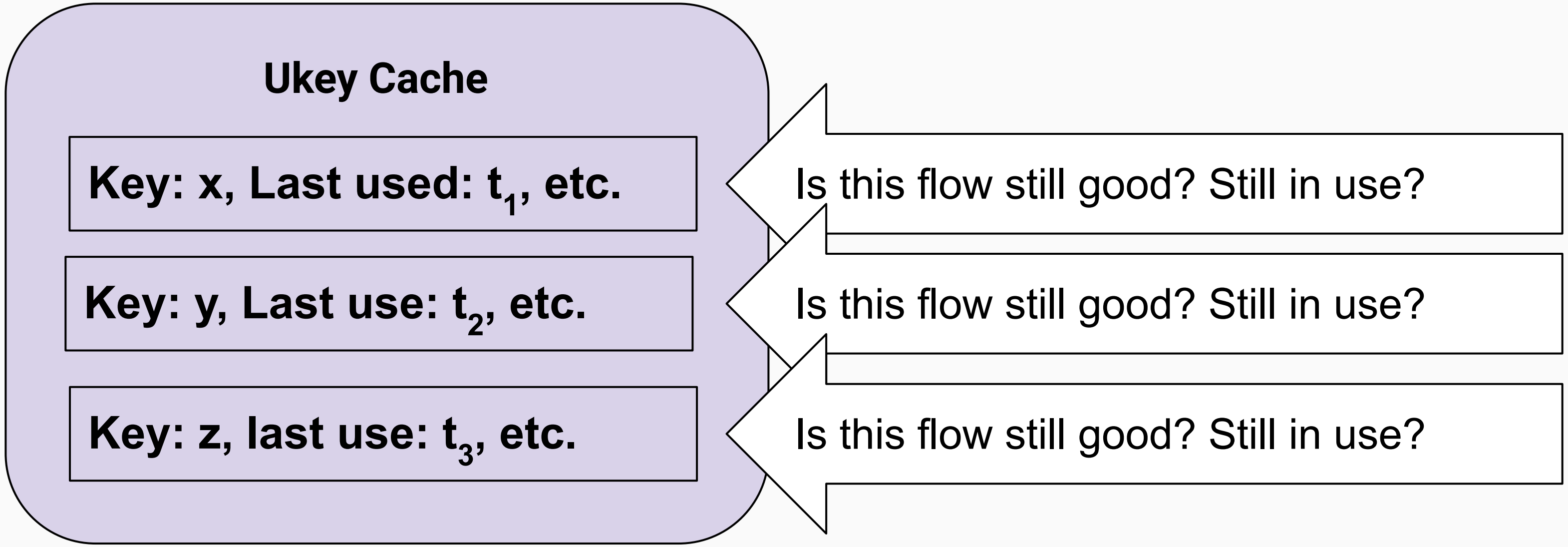
The Revalidator [3,4]

- The revalidator is Open vSwitch's garbage collector.
- Two phases: dump/mark and sweep.
    1. Each flow gets a udpif_key (ukey) containing its most recent statistics
    2. Revalidator walks through the list, "bad" flows are deleted.
- Flows are deleted silently.
- Revalidator code is in `ofproto/ofproto-dpif-upcall.c`

Dump/Mark Phase


Ukey cache
Flow: x, last used: t_1, etc.
Flow: y, last used: t_2, etc.
Key made, Flow added
Flow: z, last used: never, etc.

Sweep Phase


Ukey Cache
Key: x, Last used: t_1, etc.
Key: y, Last use: t_2, etc.
Key: z, last use: t_3, etc.
Is this flow still good? Still in use?
Is this flow still good? Still in use?
Is this flow still good? Still in use?

Why add these probes?

- Tools existed to know about flow creation.
- No tools for flow deletion.
- No way to know when/why a flow was deleted.

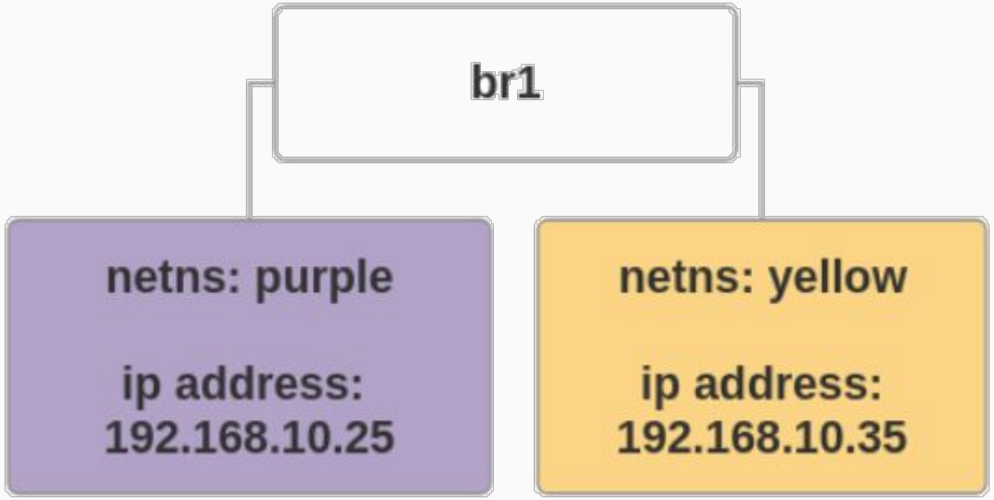
How did I implement this?

1. Installed patch that used syslogs to monitor revalidator
2. Used this to create initial USDT probes and scripts
3. Worked with Aaron to refine information captured
4. eBPF and structs are a bit tricky
    - Can’t just do `#include “ofproto-dpif-upcall.h”`
5. Created scripts to watch revalidator deletions

```
OVS_USDT_PROBE(revalidate, flow_result, reason, udpif, ukey);
```

An Example

- How many flows are created/deleted when we run each series of commands?
- What happens if we call dump-flows twice?
- Is this information correct?



```
ip link set veth_purple_br down
ip link set veth_purple_br up
ovs-appctl dpctl/dump-flows -m
```

```
ip link set br1 down
ip link set br1 up
ovs-appctl dpctl/dump-flows -m
```

```
ip link set ovs-system down
ip link set ovs-system up
ovs-appctl dpctl/dump-flows -m
```

Results

Quick Demo

Going Further

- There's a kernel tracepoint called `ovs_dp_upcall`
- Can we write a script to watch flows from here all the way to deletion?
- Potential issues:
  - Kernel tracepoint has different fields, before UFID assignment
  - Can we map these fields to the userspace probes?

Let's try it![5]

Flows from birth to death

Demo

Future Work

- At least two new patchsets since August using eBPF probes
- Would love to explore using eBPF for more than monitoring

Questions?

Thank you for having me!

eBPF logo from ebpf.io
BCC logo from https://github.com/iovisor/bcc
[1]:
https://media.defcon.org/DEF%20CON%2029/DEF%20CON%2029%20presentations/PatH%20-%20Warping%20Reality%20-%20creating%20and%20countering%20the%20next%20generation%20of%20Linux%20rootkits%20using%20eBPF.pdf
[2]: https://ebpf.io/infrastructure
[3]: https://www.openvswitch.org/support/ovscon2014/18/1230-revaliwhat.pdf
[4]:
https://developers.redhat.com/articles/2022/10/19/open-vswitch-revalidator-process-explained
[5]: https://github.com/kevinsprague/ovscon-scripts/blob/main/trace_flows.bt