Static analysis of return code propagation

Asbjørn Sloth Tønnesen

February 2nd, 2025

Last year

Brussels / 3 & 4 February 2024 schedule News Sponsors Contact

FOSDEM 2024 / Schedule / Events / Developer rooms / Network / Flying higher: hardware offloading with BIRD

Flying higher: hardware offloading with BIRD

- **Track**: Network devroom
- **Room**: UB5.230
- **Day**: Saturday
- **Start**: 14:40
- **End**: 15:00
- **Video only**: ub5230
- **Chat**: Join the conversation!

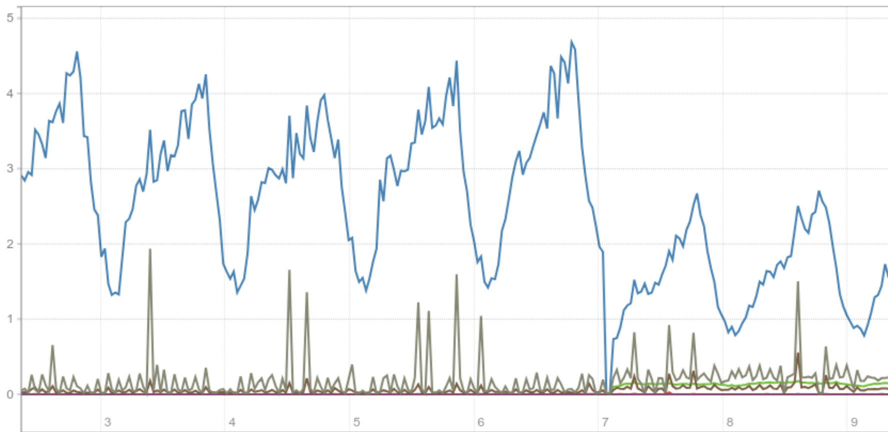
Offload IP forwarding to a SmartNIC/DPU with `tc-flower(8)`

In order to conserve CPU cycles, it can be helpful to offload all or some of the Internet routing table, to the embedded switch within modern network cards. Linux have good support for doing this through the `tc-flower(8)` API, although it was originally aimed towards OVS-offloading, it's however also capable of IP forwarding.

In the first part of the talk we will go through how `tc-flower(8)` can be used to offload IP forwarding onto a compatible SmartNIC/DPU, by scripting some `tc` commands. In the second part of the talk we will introduce `flower-route`, a new daemon, which keeps a hardware-offloaded `tc` ruleset in sync, with routing changes from a routing daemon like BIRD (or FRR, ...). Thereby attaining BGP-based IP forwarding offload.

Kernel work

- Effect of TC bypass in v6.10 (`047f340b`)



qede_parse_actions(...)

```
1  static int qede_parse_actions(...)
2  {
3      if (!flow_action_has_entries(flow_action))
4          return -EINVAL;
5      if (!flow_action_basic_hw_stats_check(flow_action, extack))
6          return -EOPNOTSUPP;
7      ...
8      return 0;
9  }
```

qede_add_tc_flower_fltr(...)

```
1  int qede_add_tc_flower_fltr(...)
2  {
3      if (qede_parse_actions(...))
4          return -EINVAL;
5      ...
6      return 0;
7  }
```

Review-ability

- I found 3 bug, of this kind, in one driver.
- 3 additional `static int` calls were fixed.
- Most had been hiding in the initial patch adding the code (2018).
- One of the bugs was activated by this tree-wide patch (`319a1d19`):

```
1  @@ -1756,6 +1757,9 @@ static int qede_parse_actions(struct qede_dev
2                   return -EINVAL;
3           }
4  
5  +        if (!flow_action_basic_hw_stats_types_check(flow_action, ext
6  +                return -EOPNOTSUPP;
7  +
8           flow_action_for_each(i, act, flow_action) {
9                   switch (act->id) {
10                  case FLOW_ACTION_DROP:
```

Return code propagation research by University of Wisconsin

- 2008: EIO: Error Handling is Occasionally Correct
- 2009: Error Propagation Analysis for File Systems
- 2010: Expect the Unexpected: Error Code Mismatches Between Documentation and the Real World
- 2011: Defective Error/Pointer Interactions in the Linux Kernel
- 2011: Finding Error-Handling Bugs in Systems Code Using Static Analysis
- Research performed by Haryadi S. Gunawi, Cindy Rubio González, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Ben Liblit at University of Wisconsin.

Intro

- `sparse` is a C language semantic parser.
- Designed to be “small and simple” and “easy to use”.
- Main `sparse` binary is a static analyser.
- Written for use with the Linux kernel, by Linus in 2003.
- Multiple other binaries `c2xml`, `cgcc`, `graph`, etc.
- Multiple `test-*` binaries, for example `test-linearize` and `test-unssa`.
- MIT licensed.

initial commit

```
1  commit 3ece2ef7c0a3d5975f65aa09911e1944e4125c45
2  Author: Linus Torvalds <torvalds@home.transmeta.com>
3  Date:   Thu Mar 13 12:53:56 2003 -0700
4
5      Yaah. I'm a retard, but I want to at least try to see how hard
6      it is to do a semantic parser that is smaller than gcc is.
7
8      Right now this is just the lexer, though, along with a test
9      app to print the results back out to verify the thing.
```

Usage with the Linux kernel

- `make CHECK=sparse all`
- Highly interlinked memory structures.
- `Kbuild` calls `$CHECK` once, per source file.

qede_add_tc_flower_fltr(...) as basic blocks

```
1  basic block .L1937
2          call.32      %r3730 <- qede_parse_actions, %arg1, %arg2, $0
3          cbr          %r3730, .L1948, .L1949
4
5  basic block .L1948
6          copy.32      %r4057 <- $0xffffffea
7          br           .L1918
8
9  basic block .L1949
10         copy.32      %r4057 <- $0
11         br           .L1918
12
13 basic block .L1918
14         ret.32       %r4057
```

A more complex register call

```
1  basic block .L29
2          load.32     %r72 <- 0[%r68]
3          cbr         %r72, .L28, .L16
4
5  basic block .L28
6          load.32     %r75 <- 0[%r65]
7          load.32     %r77 <- 4[%r65]
8          add.32      %r78 <- %r77, $192
9          load.32     %r82 <- 16[%r72]
10         call        %r82, %r75, %r78
11         br          .L16
```

Methodology and current state

- Serializing and deserializing
- Resolving call graph, across files.
- Finding entry points (`*_ops` structs)
- Set of return values per function.
- Minimizing false positives.

Future updates

- Expecting to publish kernel-wide scans within a few weeks on https://2e8.dk/rccheck/, including code for reproducing.
- Will properly do an update at BornHack 2025.