XDP BULK PACKET PROCESSING

Maciej Fijałkowski

Linux Plumbers Conference, Lisbon 2019

Legal Disclaimers

Intel technologies features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps. Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

Intel, the Intel logo, Intel. Experience What's Inside, the Intel. Experience What's Inside logo, Intel Xeon Phi, and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

*Other names and brands may be claimed as the property of others.

Problem statement

How can we apply batching to XDP?

Performance results

action	XDP_DROP	XDP_TX	XDP_REDIRECT
Perf boost	16%	70%	91%

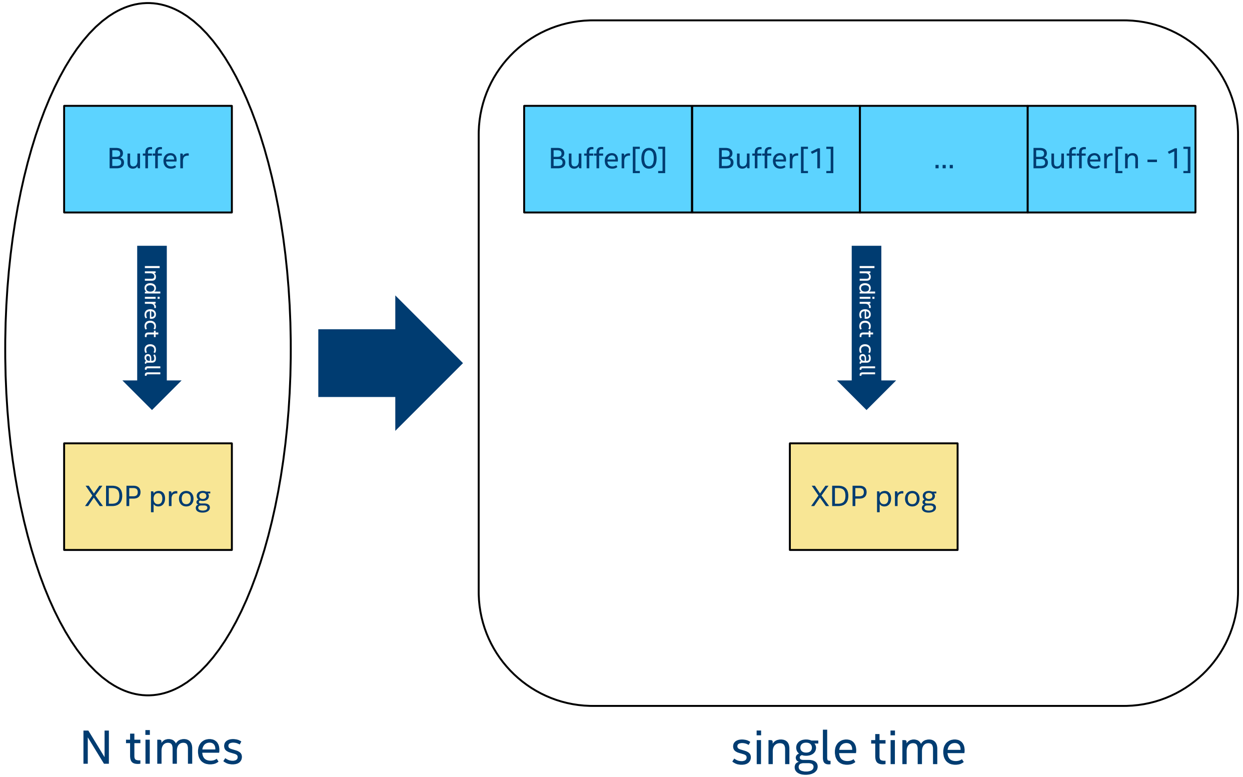
Conducted on FVL 10G, i40e driver. No fancy performance settings, just 5.3 kernel with CONFIG_RETPOLINE=y

Agenda

- Source of performance improvements
- PoC contents
  - Driver changes
  - eBPF verifier changes
- Things to be solved/questions/thoughts

Source of performance improvements

- Bulking
- Less indirect calls



DRIVER CHANGES

Storing XDP program result per XDP buffer

- Extend struct xdp_buff

```
struct xdp_buff {
     void *data;
     void *data_end;
     void *data_meta;
     void *data_hard_start;
     unsigned long handle;
     struct xdp_rxq_info *rxq;
+    unsigned int act;
 };
```

- Pass array to be filled via argument to XDP program

```
ret =
 (*(prog)->bpf_func)(ctx, (prog)->insni,
+                   results, size);
```

Storing XDP program result per XDP buffer

- Extend struct xdp_buff

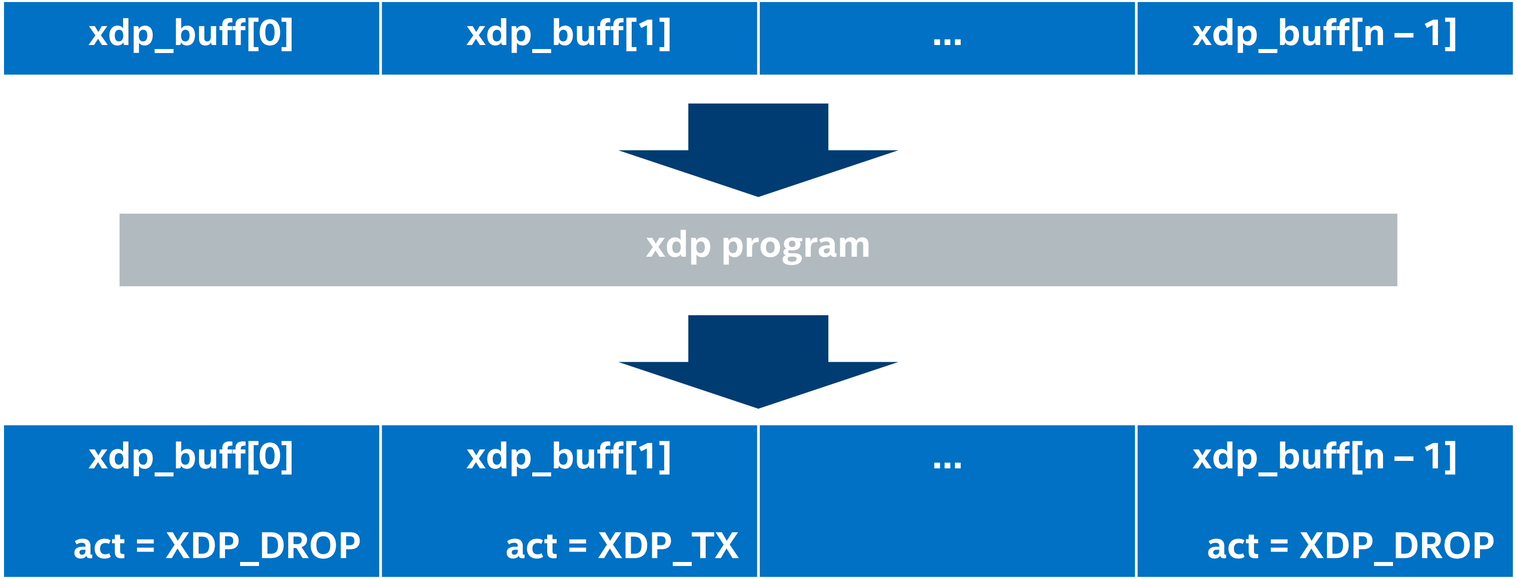
```
struct xdp_buff {
     void *data;
     void *data_end;
     void *data_meta;
     void *data_hard_start;
     unsigned long handle;
     struct xdp_rxq_info *rxq;
+    unsigned int act;
 };
```

- Pass array to be filled via argument to XDP program

```
ret =
 (*(prog)->bpf_func)(ctx, (prog)->insni,
+                    results, size);
```

Bulking in driver

```
During ring alloc, for each ring:
     vsi->rx_rings[i]->xdp_buffs = (struct xdp_buff *)kcalloc(64, sizeof(struct xdp_buff),
                                                               GFP_KERNEL);
```



xdp_buff[0] | xdp_buff[1] | … | xdp_buff[n – 1]

xdp program

xdp_buff[0] act = XDP_DROP | xdp_buff[1] act = XDP_TX | … | xdp_buff[n – 1] act = XDP_DROP

Bulking in driver – simplified pseudo code

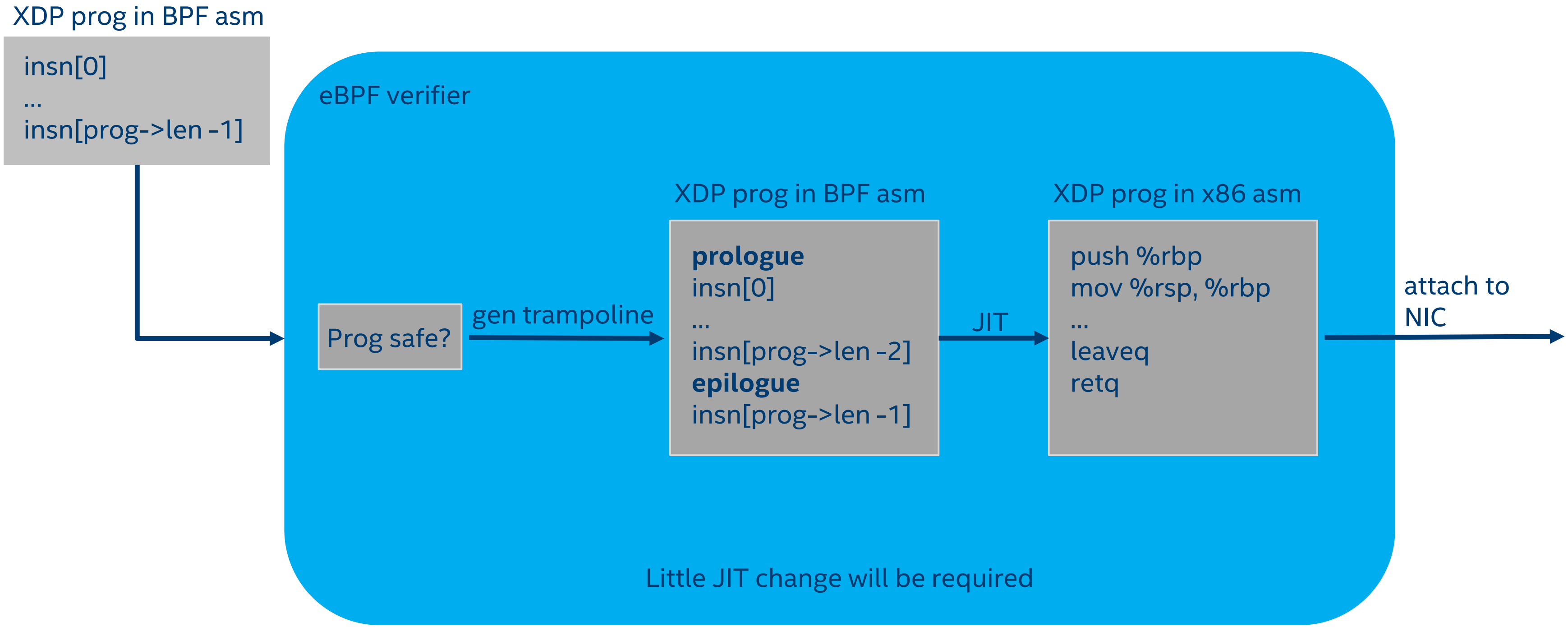
```
Clean rx interrupt:
    struct xdp_buff *xdp;
    total_rx_pkts = 0;

    while (total_rx_pkts < budget) {
        get Rx descriptor from rx_ring
        xdp = &rx_ring->xdp_buffs[total_rx_pkts];
        setup xdp_buff;
        total_rx_pkts++;
    }
```

```
    (void)bpf_prog_run_xdp(xdp_prog, rx_ring->xdp_buffs);
```

```
    for (i = 0; i < total_rx_pkts; i++) {
        xdp = &rx_ring->xdp_buffs[i];
        based on xdp->act, take appriopriate action;
    }
```

EBPF CHANGES

Trampoline patching flow



XDP prog in BPF asm

insn[0]
…
insn[prog->len -1]

eBPF verifier

Prog safe?

gen trampoline

XDP prog in BPF asm

prologue
insn[0]
…
insn[prog->len -2]
epilogue
insn[prog->len -1]

JIT

XDP prog in x86 asm

push %rbp
mov %rsp, %rbp
…
leaveq
retq

attach to
NIC

Little JIT change will be required

insn – a single BPF assembly instruction

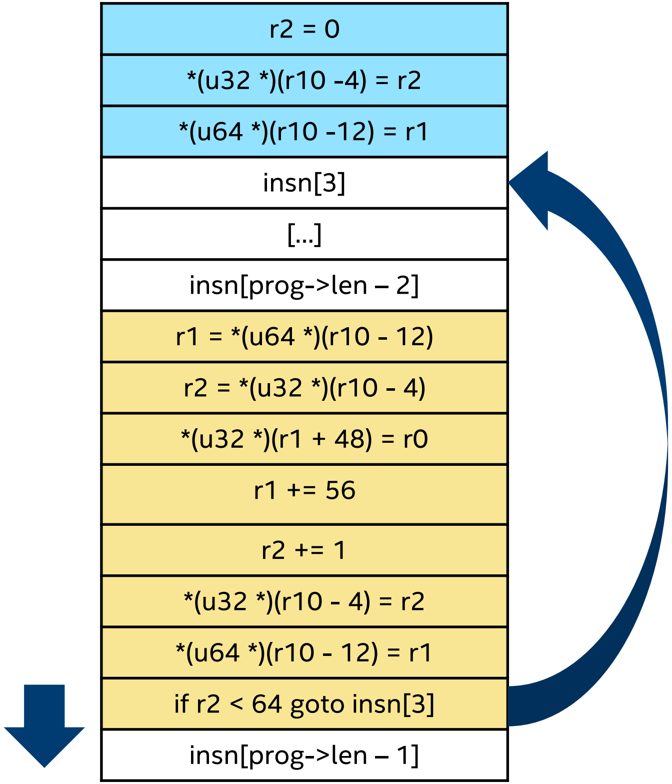
eBPF calling convention

Before we dive into eBPF, a little reminder how calling convention is defined:

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

Taken from https://www.kernel.org/doc/Documentation/networking/filter.txt

eBPF program layout after generating trampoline



```
r2 = 0
*(u32 *)(r10 -4) = r2
*(u64 *)(r10 -12) = r1
insn[3]
[...]
insn[prog->len – 2]
r1 = *(u64 *)(r10 - 12)
r2 = *(u32 *)(r10 - 4)
*(u32 *)(r1 + 48) = r0
r1 += 56
r2 += 1
*(u32 *)(r10 - 4) = r2
*(u64 *)(r10 - 12) = r1
if r2 < 64 goto insn[3]
insn[prog->len – 1]
```

Prologue is executed once, whilst epilogue is executed on each loop iteration

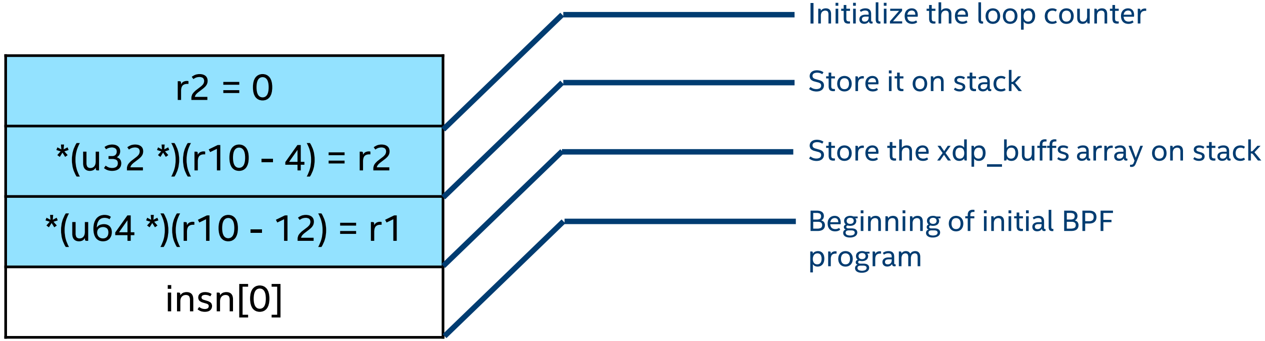
prologue

Initial program insns

epilogue

eBPF trampoline prologue section

Simple as that:



r2 = 0

*(u32 *)(r10 - 4) = r2

*(u64 *)(r10 - 12) = r1

insn[0]

Initialize the loop counter

Store it on stack

Store the xdp_buffs array on stack

Beginning of initial BPF program

At the start of a program, R1 is of a PTR_TO_CTX register type.
This means that, for XDP case, it is holding the xdp_buff pointer that was initialized by the network driver that is running the XDP program against that buffer.

eBPF trampoline prologue section, continued

Since we're consuming 12 stack bytes, we need to refresh the instructions that are making use of stack in initial program.

There are two cases that need to be handled:

-= 12

- store/load onto/from R10, e.g.:
  - `BPF_LDX_MEM(BPF_DW, BPF_REG_2, BPF_REG_10, -12),`
- ALU ops on PTR_TO_STACK register types, e.g.:
  - `BPF_MOV64_REG(BPF_REG_1, BPF_REG_10),`
  - `BPF_ALU64_IMM(BPF_ADD, BPF_REG_1, -20),`

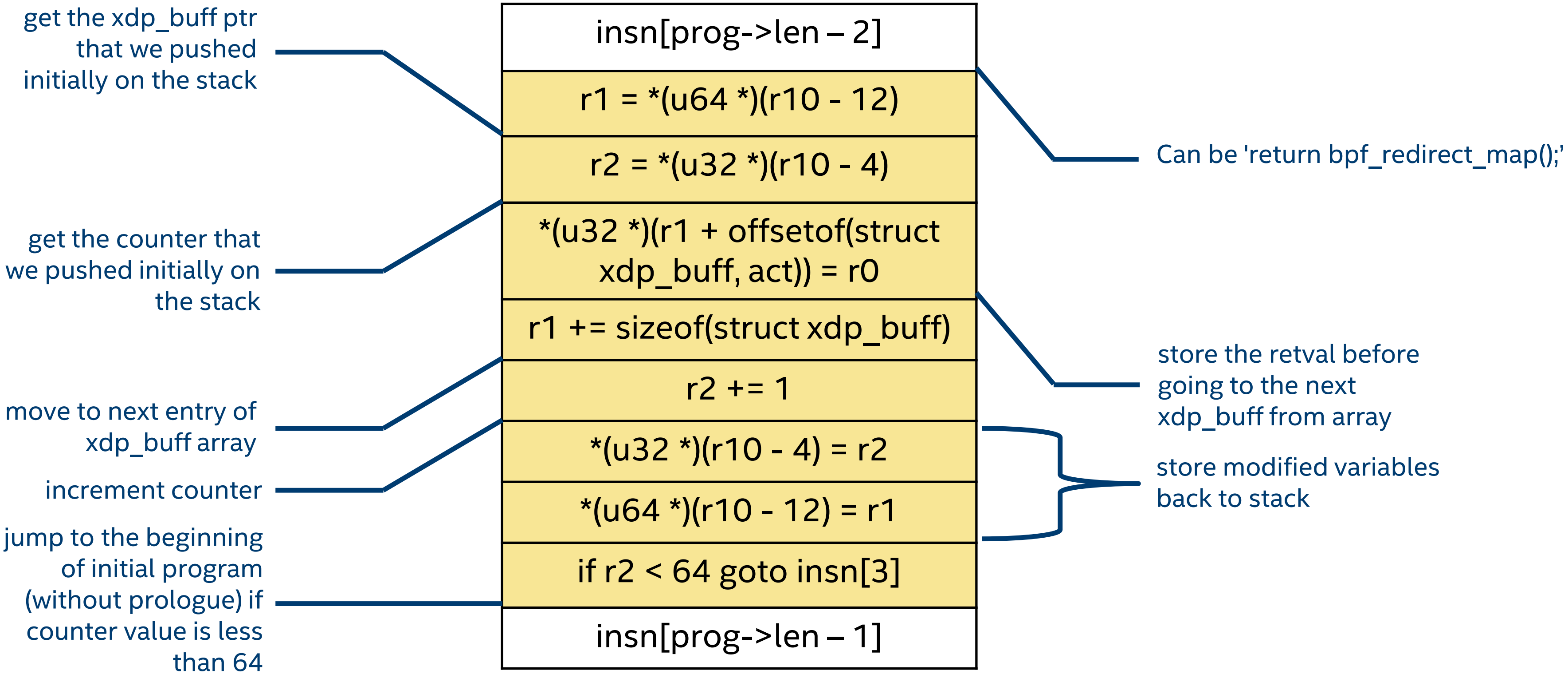
eBPF trampoline prologue section, continued

In JIT generation, the additional 12 bytes needs to be taken into account when stack depth is looked up

```
   0:    push    %rbp
   1:    mov     %rsp,%rbp
   4:    sub     $0x10,%rsp
   b:    push    %rbx
   c:    push    %r13
   e:    push    %r14
  10:    push    %r15
```

Otherwise, caller’s (driver’s) stack variables might get overwritten.

eBPF trampoline epilogue section



```
insn[prog->len – 2]
r1 = *(u64 *)(r10 - 12)
r2 = *(u32 *)(r10 - 4)
*(u32 *)(r1 + offsetof(struct
xdp_buff, act)) = r0
r1 += sizeof(struct xdp_buff)
r2 += 1
*(u32 *)(r10 - 4) = r2
*(u64 *)(r10 - 12) = r1
if r2 < 64 goto insn[3]
insn[prog->len – 1]
```

get the xdp_buff ptr that we pushed initially on the stack

get the counter that we pushed initially on the stack

move to next entry of xdp_buff array

increment counter

jump to the beginning of initial program (without prologue) if counter value is less than 64

Can be 'return bpf_redirect_map();'

store the retval before going to the next xdp_buff from array

store modified variables back to stack

Things to be solved/questions/thoughts

1. “prefetch” instruction in BPF asssembly
2. Selftests
3. How to provide backward compatibility?
4. Sort actions?
5. How much AF_XDP would benefit from it?
6. Thought – driver changes ARE required
7. Thought – boost for Tx/Redirect speaks for itself

Q&A

BACKUP

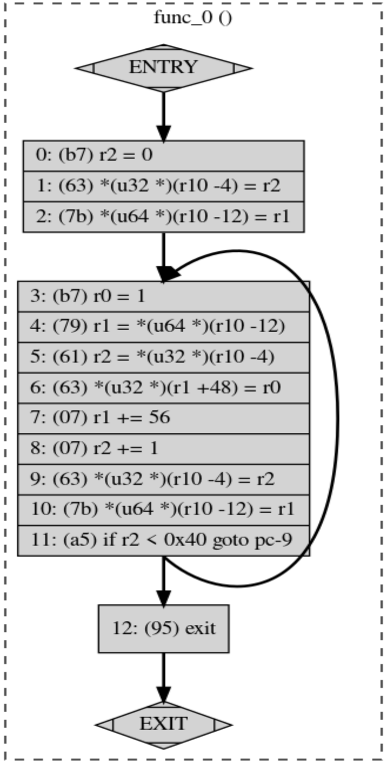
eBPF program layout after generating trampoline
putting it all together

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

__section("prog")
int xdp_drop(struct xdp_md *ctx)
{
    return XDP_DROP;
}
```

Clang compiler

```
R0 = 1
exit
```

Generate trampoline



```
0: (b7) r2 = 0
1: (63) *(u32 *)(r10 -4) = r2
2: (7b) *(u64 *)(r10 -12) = r1
3: (b7) r0 = 1
4: (79) r1 = *(u64 *)(r10 -12)
5: (61) r2 = *(u32 *)(r10 -4)
6: (63) *(u32 *)(r1 +48) = r0
7: (07) r1 += 56
8: (07) r2 += 1
9: (63) *(u32 *)(r10 -4) = r2
10: (7b) *(u64 *)(r10 -12) = r1
11: (a5) if r2 < 0x40 goto pc-9
12: (95) exit
```

JIT

```
0xffffffffc04bea37:
   0:   push   %rbp
   1:   mov    %rsp,%rbp
   4:   sub    $0x10,%rsp
   b:   push   %rbx
   c:   push   %r13
   e:   push   %r14
  10:   push   %r15
  12:   pushq  $0x0
  14:   xor    %esi,%esi
  16:   mov    %esi,-0x4(%rbp)
  19:   mov    %rdi,-0xc(%rbp)
  1d:   mov    $0x1,%eax
  22:   mov    -0xc(%rbp),%rdi
  26:   mov    -0x4(%rbp),%esi
  29:   mov    %eax,0x30(%rdi)
  2c:   add    $0x38,%rdi
  30:   add    $0x1,%rsi
  34:   mov    %esi,-0x4(%rbp)
  37:   mov    %rdi,-0xc(%rbp)
  3b:   cmp    $0x40,%rsi
  3f:   jb     0x000000000000001d
  41:   pop    %rbx
  42:   pop    %r15
  44:   pop    %r14
  46:   pop    %r13
  48:   pop    %rbx
  49:   leaveq
  4a:   retq
```