OPENED Tool for Managing eBPF Heterogeneity

Microservices Observatory (microserviceobservatory.github.io)

Theophilus A. Benson  Brown University  tab@cs.brown.edu
Palanivel Kodeswaran  IBM Research  palani.kodeswaran@in.ibm.com
Sayandeep Sen  IBM Research  sayandes@in.ibm.com
One Off Programs

Complex Codebase(s)

Observability

Network Functions

*Code from a Katran function
Implications of Monolith on Developer Productivity

Developing a new program

Find sub functionality on GitHub

Extracting and reusing functionality is non-trival

Step 1: Extract lines

Step 2: Identify + Extract Deps

 Surprise Step 3: Rewrite for your target hookpoint
The OPENED Vision

- Select eBPF function of interest
- Extract Function
- Transform Function
- Extends/Builds on Transformed Function

eBPF Developer
OPENED Framework
eBPF Developer
OPENED Vision: Reduce time to new functionality development

- Automated extraction of relevant code

- Automated transformation of code
  - Enable moving code between hook-points
  - Enable moving code between programs

- Developer-first automation
  - Extraction + Transformation guided by developer choices
Road Map

- Extraction
  - Transformation
  - Demo
- Call to Arms
Extraction

Extract eBPF func as an independently loadable module

- Identify all dependencies of the eBPF function
  - Dependencies: function call graph, Maps & associated structures, header files
  - Extract relevant dependencies while
    - Ensuring correctness and minimizing technical debt
- Caveat: original code must pass verification
Challenge: Dependency Extraction

Function Dependencies

- Extended codequery tool[1],
  - Recursively identify function call graph
  - Uses cscope and ctags and sqlite internally
- Used TXL source transformation tool to annotate the function definitions
- Python script to extract functions and header files

```c
SEC("decap")
int xdpdecap(struct xdp_md* ctx) {
/* Extracted from
/root/github/demo_lpc/
codequery/katran/decap_kern.c  startLine: 223
endLine: 247 */
SEC("decap")
int xdpdecap(struct xdp_md* ctx) {
```
Challenge: Dependency Extraction

Map Definitions

- eBPF specific method of tracking
  bpf_map_update/lookup_elem while parsing call flow graph
- TXL code transformation tool to annotate maps and other data structures needed
- Python script to extract structures

```c
bpf_map_lookup_elem(&decap_dst, ...)
bpf_map_update_elem(lru_map, ...);
```

```
extracted.c
...
Struct lru_map{};
Struct decap_dst{}
...
```
Challenge: Dependency Extraction

Multiple declaration of dependencies (both maps & functions)

```plaintext
{#funcName,count,[FileName,lineNumber]}
.....
ncrement_quic_cid_version_stats,1,[<dir...>/balancer_kern.c,445]
ncrement_quic_cid_drop_no_real,1,[<dir...>/balancer_kern.c,460]
process_l3_headers,2,[<dir...>/balancer_kern.c,158],[<dir...>/decap_kern.c,34]
ncrement_quic_cid_drop_real_0,1,[<dir...>/balancer_kern.c,470]
process_encaped_ipip_pckt,2,[<dir...>/balancer_kern.c,340],[<dir...>/decap_kern.c,85]
parse_udp,1,[<dir...>/pckt_parsing.h,76]
REPORT_PACKET_TOOBIG,2,[<dir...>/introspection.h,32],[<dir...>/introspection.h,40]
.....
```
## Preserve MACRO Definitions during Extraction

Identify and propagate preprocessor guards into extracted code

<table>
<thead>
<tr>
<th>balancer_kern.c</th>
</tr>
</thead>
</table>
| ```c
  #ifdef GLOBAL_LRU_LOOKUP
  __attribute__((__always_inline__)) static inline bool
  reals_have_same_addr(
    struct real_definition* a,
    struct real_definition* b) {
    ...
    ...
  }
  __attribute__((__always_inline__)) static inline int
  perform_global_lru_lookup(
    struct real_definition** dst,
    struct packet_description* pckt,..){
    ...
    ...
  }
  #endif // GLOBAL_LRU_LOOKUP
``` |

<table>
<thead>
<tr>
<th>extracted.c</th>
</tr>
</thead>
</table>
| ```c
  #ifdef GLOBAL_LRU_LOOKUP
  /* Extracted from balancer_kern.c startLine: 261 endLine: 277 */
  __attribute__((__always_inline__)) static inline bool
  reals_have_same_addr(
    struct real_definition* a,
    struct real_definition* b) {
    ...
    ...
  }
  #endif
  #ifdef GLOBAL_LRU_LOOKUP
  /* Extracted from balancer_kern.c startLine: 279 endLine: 337 */
  __attribute__((__always_inline__)) static inline int
  perform_global_lru_lookup(
    struct real_definition** dst,
    struct packet_description* pckt,..){
    ...
    ...
  }
  #endif // GLOBAL_LRU_LOOKUP
``` |
Challenge: Minimize Code Debt

- Maintain ordering between definitions and invocations
- Propagate license into newly created c file (with extracted code).
- Identify and copy relevant current directory includes into extraction site.
  - Introduce preprocessor guards in new header files
- Rewrite Makefiles (currently Manual).

```c
#include balancer_const.h
#ifndef BALANCER_CONST_OPF
#include balancer_const.h
#endif
```
Road Map

- Extraction
- Transformation
- Demo
- Call to Arms
Implications of Monolith on Developer Productivity

Developing a new program

Find sub functionality on GitHUB

Extracting and reusing functionality in non-trivial

Step 1: Extract lines

Step 2: Identify + Extract Deps

Surprise Step 3: Rewrite for your target Hookpoint
Nuances of Hookpoint Transformation

- Code written for one hookpoint does not port to another trivially
  - Different Header files, Actions, Information source & Helper functions
- Some capabilities are hookpoint specific, e.g., bpf_redirect_maps
- However, some capabilities are overlapping, e.g., access to 5-tuple
  - Even if expressed differently [They are portable]

*Talk to us for more transformation use cases*
Our Solution for Hookpoint Transformation

- Code written for one hookpoint does not port to another trivially
  - Different Header files, Actions, Information source & Helper functions
- Some capabilities are hookpoint specific, e.g., bpf_redirect_maps
- However, some capabilities are overlapping, e.g., access to 5-tuple
  - Even if expressed differently [They are portable]
- Database of domain specific functionality mapping between hookpoints
  - Currently working only for XDP ⇒ TC
  - Transformation rules written in using Coccinelle and TXL
- Report error, if transformation is unknown

*Talk to us for more transformation use cases
Transformation*

- Porting Header files is trivial
  - Include/Exclude headers e.g.,
    `#include <linux/pkt_cls.h>` for XDP⇒TC

- Porting Actions is straightforward, E.g.,
  - DROP and PASS are transformable
    - XDP_DROP ⇒ TC_ACT_SHOT
    - XDP_PASS ⇒ TC_ACT_OK
  - XDP_TX is hookpoint specific, and does not port, report Error

*Talk to us for more transformation use cases

```
rule replaceXDP_DROP
  replace [token]
  XDP_DROP
  by
  TC_ACT_SHOT
end rule
```

```
rule replaceXDP_PASS
  replace [token]
  XDP_PASS
  by
  TC_ACT_OK
end rule
```

TXL Rule snippet
Porting information source is straightforward,

- (If available) Replace with information source
  - [XDP] `eth->h_proto` ⇒ `ctx->protocol` [TC]
  - [XDP] `vlan_hdr->h_vlan_TCI` ⇒ `ctx->vlan_tci` [TC]

```c
@replaceethproto@
identifier p, c, fn; 
type t;
struct ethhdr *e;
@@
t fn(struct __sk_buff *ctx){
    ...
    - e->h_proto
    + ctx->protocol
    ...
}
```

Coccinelle Rule snippet

*Talk to us for more transformation use cases*
Transformation*

Porting helper functions is non-trivial

- **Simple:** static transformation rules, E.g.,
  - `bpf_redirect()` ports from XDP to TC with FLAG set to ingress

- **Complex:** developer must introduce new rules based on intended use
  - `bpf_xdp_adjust_head(E1,E2) ⇒ bpf_skb_adjust_room(E1,E2, BPF_ADJ_ROOM_MAC, BPF_F_ADJ_ROOM_ENCAP_L3_IPV4/IPV6)`

*Talk to us for more transformation use cases*
Current Prototype

- Extraction: 1448-LoC
  - 251 LoC TXL\(^1\) (Grammar specification)
  - 547 Extended Codequery\(^3\)
- Transformation: 200-LoC
  - 74 LoC in Coccinelle\(^2\)
  - 68 LoC in TXL
- Extracted and transformed functions within Meta’s Katran, Mizar, Suricata, Cloudflare’s XDP\_drop

\(^{1}\) [http://txl.ca/](http://txl.ca/)
\(^{2}\) [https://coccinelle.gitlabpages.inria.fr/](https://coccinelle.gitlabpages.inria.fr/)
\(^{3}\) [https://github.com/ruben2020/codequery](https://github.com/ruben2020/codequery)
Road Map

● Extraction

● Transformation

Demo

● Call to Arms
Road Map

- Extraction
- Transformation
- Demo

Call to Arms
Future Plans

- Decompose convert open source programs into L3AF/Polycube/BPFD modules
- Expand the set of supported transformation rules
- Improve usability of our framework
Join the OPENED Community (DevTools for Supporting Modular eBPF Programs)

Join Us!
Submit your use cases for programs to be decomposed

Microservices Observatory (microserviceobservatory.github.io)