Ahead-of-time compiled bpftrace programs

Daniel Xu

Current compilation model

```
# bpftrace -e 'BEGIN { print("hello world") }'
```

- Bytecode built from scratch for *every* invocation, then run
- “Duplicated” work between identical invocations:
  - Lexing, parsing, semantic analysis, field analysis, codegen (modulo offsets, constants, and kernel features)
- “Unique” work “...”:
  - Tracepoint format parsing, clang parsing of headers and/or reconstructed BTF definitions, runtime setup

Current compilation model cont.

- Advantages
  - Straightforward to implement -- any host information you need during compilation can be trivially accessed
  - Struct layouts, symbol addresses, tracepoint fields, etc. are more likely to be accurate/up-to-date
- Disadvantages
  - Shipping and running LLVM on production hosts is heavy and expensive (size + memory + cpu)
  - Time to first trace is slow b/c we have to run all AST passes and runtime setup
  - Difficult to sign bpftrace scripts

Why AOT?

- Smaller binaries
  - -rwxr-xr-x 1 root root 99M Sep 7 17:26 /usr/lib/libLLVM-12.so
- Faster
- Have cake and eat it too
  - Ease of use along with efficiency of libbpf-based tools

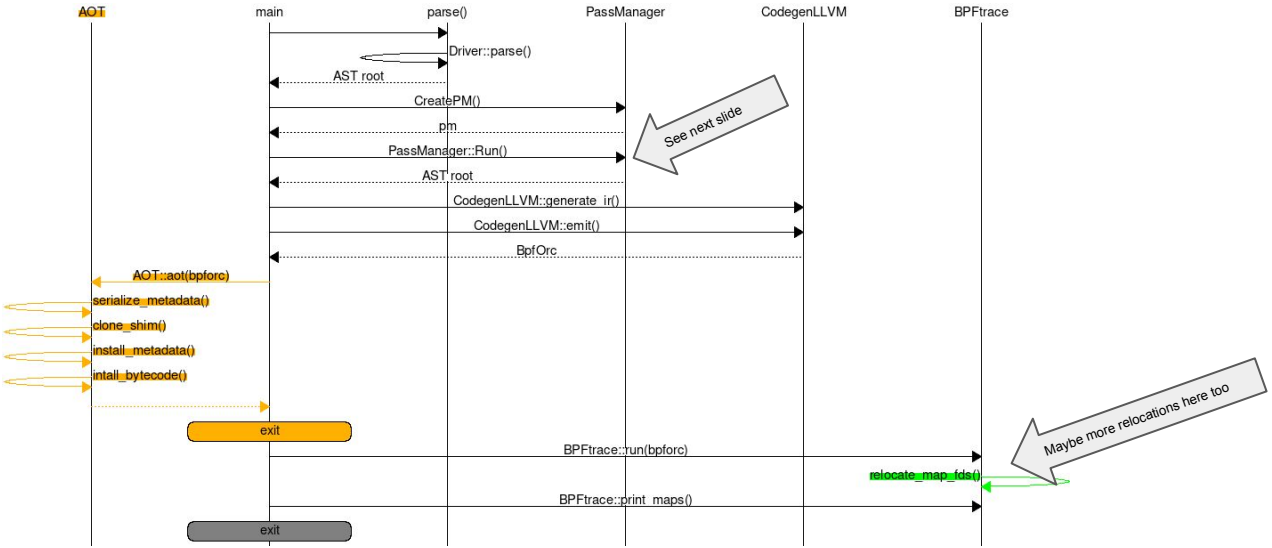
AOT end goal

```
# bpftrace -e 'BEGIN { print("hello world") }' --aot out.btaot

# file ./out.btaot
./out.btaot: ELF 64-bit LSB pie executable, x86-64,[...]

# ldd ./out.btaot
    statically linked

# ./out.btaot
Attaching 1 probe...
hello world
```

AOT design

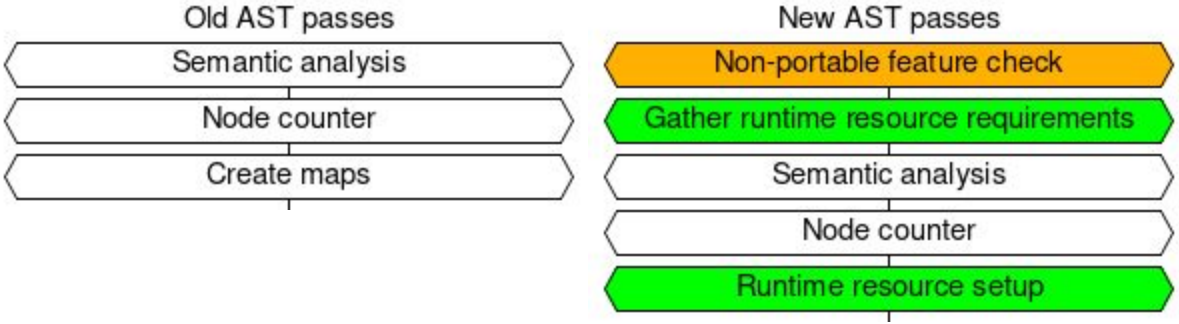
- Ship a fully executable runtime shim with bpftrace
- When compiling a AOT bpftrace program:
  - Build the metadata
  - Build the bytecode
  - Make a copy of runtime shim and store metadata + bytecode into a special ELF section (this is the final executable)
- When the shim runs, it knows to look inside itself for the metadata + bytecode and start execution

AOT design cont.



- Green -> added to all codepaths
- Orange -> added to AOT compile codepath

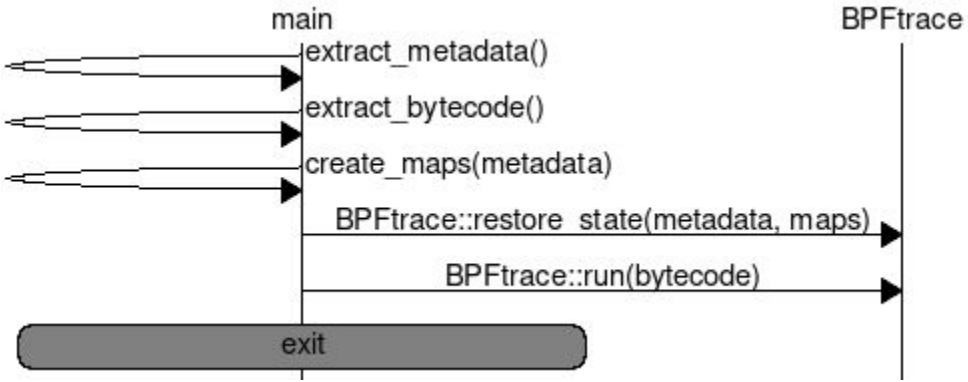
AOT design cont.



- Green -> added to all codepaths
- Orange -> added to AOT compile codepath

AOT design cont.

AOT execution



Solved problems

- Hard coded map FDs in bytecode
  - Create maps after codegen and fixup immediates before loading prog
- How to handle async IDs
  - Ship all the (format string, args) with the metadata
- How to drop LLVM + libclang dependency for AOT binaries
  - New bpftrace AOT runtime target (aot-rt) target
  - New bcc .so: https://github.com/iovisor/bcc/pull/3516
  - Not completely solved!
- How to disable (currently) unsupported features
  - New feature check AST pass on compile path

Unsolved problems

- How to create a fully static aot-rt binary?
  - Can cmake be finessed enough to accomplish this?
  - What about BCC, libc, etc.?

Unsolved problems cont.

- How to relocate field accesses on kernel structs?
  - Reuse libbpf CO-RE or roll custom solution?
- Is using libbpf for loading feasible?
  - bpftrace code modifications
  - Ability to mock out maps / unit testing?
  - Extra overhead constructing ELF files for libbpf to work with?
  - Tension between distro packaging rules and vendoring

Unsolved problems cont.

- How to completely remove dependence on LLVM for AOT binary?
  - Currently using LLVMDemangle.a and LLVMSupport.a for C++ symbol demangling
  - C++ provided demangler does not work as well

Unsolved problems cont.

- What to do about tracepoint fields?
  - Assume they’re stable enough between hosts/kernels?

Unsolved problems cont.

- Other disabled features
  - Positional parameters
  - curtask builtin
  - kaddr()/uaddr()/cgroupid() builtins
  - USDTs
    - Requires analyzing binary for precise offset and argument information
  - watchpoints

Current status

- Functional prototype merged into master
- Metadata + bytecode currently separate file (not embedded into aot-rt)
- Real runtime test checked into CI:

```
NAME profile
RUN {{BPFTRACE}} -e 'profile:hz:599 { @[tid] = count(); exit();}' --aot \
    /tmp/tmpscript.btaot && {{BPFTRACE_AOTRT}} /tmp/tmpscript.btaot
EXPECT \@\[[0-9]*\]\:\s[0-9]
TIMEOUT 5
```

Questions?