Cross-platform BPF compiler issues

Dave Thaler

Multiple compilers, multiple runtimes

Compilers:

- LLVM/clang, gcc, other backends from rust or other compilers?

Runtimes:

- Linux kernel, eBPF for Windows, uBPF, rbpf, hBPF, bpftime, offload cards (e.g., Netronome), etc.

Goal:

allow using a compiler with any BPF-compliant runtime

What does “compliant” mean?

- draft-ietf-bpf-isa defines “conformance groups”
  - Logical units of functionality, where a runtime conforms to a set of groups
  - Only “base32” is required

Name	Description	Includes	status
atomic32	32-bit atomic instructions	-	Permanent
atomic64	64-bit atomic instructions	atomic32	Permanent
base32	32-bit base instructions	-	Permanent
base64	64-bit base instructions	base32	Permanent
divmul32	32-bit division, multiplication, and modulo	-	Permanent
divmul64	64-bit division, multiplication, and modulo	divmul32	Permanent
packet	Legacy packet instructions	-	Historical

Example: add some instructions to “example”

Conformance groups:

name	description	includes	excludes	status
example	Example instructions	-	-	Permanent

Instructions:

opcode	...	description	groups
aaa	...	Example instruction 1	example
bbb	...	Example instruction 2	example

Example: add some instructions to “example”

Conformance groups:

name	description	includes	excludes	status
example	Example instructions	-	-	Permanent
examplev2	**Newer set of example instructions**	**example**	**-**	**Permanent**

Instructions:

opcode	...	description	groups
aaa	...	Example instruction 1	example
bbb	...	Example instruction 2	example
ccc	**...**	**Example instruction 3**	**examplev2**
ddd	**...**	**Example instruction 4**	**examplev2**

Example: deprecate some instrs in “example”

Conformance groups:

name	description	includes	excludes	status
example	Example instructions	-	-	Permanent

Instructions:

opcode	...	description	groups
aaa	...	Good example instruction 1	example
bbb	...	Good example instruction 2	example
ccc	...	Bad example instruction 3	example
ddd	...	Bad example instruction 4	example

Example: deprecate some instrs in “example”

Conformance groups:

name	description	includes	excludes	status
example	Example instructions	-	-	Permanent
legacyexample	**Legacy example instructions**	**-**	**-**	**Historical**
examplev2	**Example instructions**	**example**	**legacyexample**	**Permanent**

Instructions:

opcode	...	description	groups
aaa	...	Good example instruction 1	example
bbb	...	Good example instruction 2	example
ccc	...	Bad example instruction 3	example**, legacyexample**
ddd	...	Bad example instruction 4	example**, legacyexample**

Impact on runtimes and compilers

- A runtime conforms to a set of conformance groups
  - Linux: base64, atomic64, divmul64, packet (plus groups those include)
- Other runtimes might have a different list
  - E.g., an offload card that supports only 32-bit conformance groups
- Any new instructions require newly named conformance groups that should get registered
- Each runtime is responsible for documenting what conformance groups it supports

Impact on compilers

- Compilers should allow specifying a set of conformance groups
  - Ok to default to the set that Linux supports if desired
  - Using “cpu versions” for BPF is historical
- Might be specified using deltas, or a full list, with some default (e.g., all current groups but packet)
- Runtime that supports packet and some future group (e.g., “callx”):
  - Delta: `--include_groups packet,callx`
  - Full: `--groups base64,divmul64,atomic64,packet,callx`
- Runtime without atomics:
  - Delta: `--exclude_groups atomic32`
  - Full: `--groups base64,divmul64`
- Runtime without 64-bit instructions:
  - Delta: `--exclude_groups base64,divmul64,atomic64`
  - Full: `--groups base32,divmul32,atomic32`

psABI issues

- How many BPF registers are there?
- Which ones are scratch vs saved across calls?
- Which register is the stack pointer?
- How large is the stack?
- How much stack space does a bpf2bpf callee get?

- Compiler has to either:
  - Only support one psABI and thus only runtimes that use that one
  - Have a way to specify which psABI to generate code for

Verifier issues

- Different runtimes may have different verifiers
  - Linux kernel verifier, PREVAIL, possibly others
- Some compiler optimizations may not work with all verifiers
  - E.g., PREVAIL collapses joins for scalability and so doesn't support correlated branches sometimes generated in LLVM>11 with –O2
  - See Alan's talk earlier today
- Compiler optimizations might be independent of target (BPF) and so taking BPF-*runtime* specific data into account is even harder
  - Probably fine if optimizations can be enabled/disabled at some level of granularity, whether by command line or by code pragmas etc.
  - Hard if all-or-nothing like -O2