data-motion

An optimizing intermediate representation
for packet processing programs

The case for an IR

- Many source DSLs for packet processing:
  - P4,
  - NPL,
  - C
  - Rust/Oxide
  - C++
  - And potentially new DSLs …
- Many targets:
  - Various kinds of SmartNICs
  - Programmable Switches
  - DPDK (Fabian's unnamed project)
  - Fixed-function switches (switchV, etc.)
  - WASM sandboxes in service proxies
  - eBPF sandboxes in the kernel
  - Microcode
  - WASM at the edge (Cloudflare workers, AWS@Edge)
  - Stream processing

What an IR provides?

- Common optimizations in the style of LLVM
- Lower porting effort
- Pushing developer effort “upward” from the vendor and into the open-source community
- IR specifications can be iterated upon faster than a source language:
  - Less concern about aesthetics, more focused on pragmatics

What the IR should contain?

Tentative list:

- Filtering/predicate constructs on packets
- Packet manipulation constructs
- Stateful processing: Detailed semantic models of stateful hardware units or stateful updates in software, like atoms but more general.
- Control plane variables, or control plane state more generally: Mostly immutable state.
- Resource constraints
- Parsing constructs and Glen Gibb's parser model: Model parser as a function from byte-stream to packet headers
- Chaining/composition of these language constructs (essentially to capture data flow between different elements)

Other stuff:

- First order formal semantics for all these constructs to enable verification, symbolic execution, and synthesis.
- Multiple levels of abstraction for progressive lowering to hardware—either automatically or manually.
- Minimal innovation for maximal impact: Keep the core IR extremely simple and lightweight
- Keep the IR in SSA form like LLVM and MLIR. In general seems to be a good idea.

Some use cases

- Ctrl-data-plane joint optimization via partial evaluation: treating ctrl plane as a constant (like Flay)
- New optimization techniques: e-graphs, solvers, ML, etc.
- A common IR to move code around between switches, hosts, proxies, etc: e.g, network-wide programming: https://anirudhsk.github.io/papers/sluice.pdf
- Network-wide enforcement of security policies: topology-aware access control
- Zero trust through pervasive network programmability: Similar to Hydra at Stanford, but extend it to do hop by hop checking across switches, NICs, hosts, etc.
- Network functions as a service: Network functions are programmed across different frameworks. (NetFaaS)
- Transpiling between P4 and NPL, P4 and C dialects, etc
- New network DSL for education for programming switches, hosts, etc.

Some use cases-2

- Iterative refinement from high level algorithms to low level constraints
- Fabian’s idea: A general test oracle for packet processing even beyond P4. This might turn out to be a fairly important use case. Not sure LLVM has really explored something like this.
- Stream processing use cases
- Ir primitives could be lowered into hardware primitives if sufficiently useful. This is the idea of the sparse abstract machine ASPLOS 23 paper.
- Moving around predicates for a database to different points in the network: predicate push down and predicate move around, look at the Kexin Rong paper at VLDB 23 for this stuff in related work
- The application defined networks work out of UW: we could use this IR as a common representation for what each programmable device must do for a particular application.

Measuring success

- Can we build a better P4 compiler through this IR that produces better resource usage than existing commercial ones?
  - Say for DPDK, Tofino, SmartNIC, or an FPGA.
  - Better = more optimized, more diagnostics, more tooling, etc.
- Can we bring up a new compiler faster? Either,
  - For a new source language (ref. education DSL in previous slide)
  - For a new target device
- Can we build a compiler that offers better error messages?
  - This was the main benefit of clang over gcc.
- Show equivalence to some automaton maybe like netkat and Sam's equivalence to the streaming model (ASPLOS 23)?
- Can we make eBPF compilation safer?

Other such irs

- LLVM of course
- XLA
- NVVM IR
- DLVM
- TVM
- Intel nGraph
- Weld
- Calyx
- CIRCT
- Codon IR for the Codon framework
- The Sparse Abstract Machine (SAM) for dataflow architectures (https://dl.acm.org/doi/pdf/10.1145/3582016.3582051)
    - SAM seems to be doing exactly what we are trying to do, but for sparse computing instead of packet processing; they draw an analogy to LLVM for CPUs
    - Worth understanding it in full detail
- Mihai Budiu et al.'s DBSP VLDB 23 best paper (basis of Feldera):
    - seems like an IR/algebra for stream processing.

Other domain specific compilers

- Halide
- Julia
- Pandas
- Taco
- StreamJIT

Educational uses

- As a test, develop a new educational DSL
- Use this new IR to translate this new educational DSL down to various different platforms:
  - eBPF
  - WASM
  - Tofino
  - Trident
  - Various SmartNICs
  - FPGAs

Other ideas

- Submit white paper on this for NSF nets wired workshop
- Submit ccr paper on this (but ccr visibility isn't too high)
- Apply these ideas to stream processing and
- Potentially use this as a DSL for the RPC processor work that Tao has been doing.
- Look at the work Modular is doing around the Mojo language and the modular engine. Apparently has real world speedups around 100x. These seem like the kinds of wins we should show with our IR.
- IR should be batteries included so that it applies a suite of optimizations out of the box.

Potential roadblocks

- What if the IR can’t describe a language construct? IR needs to be sufficiently expressive (and at the same time, not too bloated) for the source languages we are considering.
- Do we produce P4 (or any other language) from this IR? Or IR from this P4? Or both?
- Is the IR a superset of P4 and eBPF and intended to cover all of their expressive power?
- Do high-level constructs (and their associated semantics) get lost when we translate them into a common IR?
- Is there enough common stuff across these targets and source languages for an IR to even be useful?
- Why not directly use LLVM?

Conceptual questions

- What is the boundary of this IR?
  - Specifically, some of these ideas can be generalized all the way to dataflow architectures like Plasticine.
- What makes packet processing special at a foundational level?

Mihai’s feedback

- TODO

Nate's feedback

One way to start work

- Using Flay as an initial prototyping vehicle or use case for the idea
- Using Proseco as another use case
- Could potentially get help from an undergrad interested in research