Some IR Questions

Ben Mack-Crane

Huawei

What is required in an IR?

- Match Action Table definitions
- Data definitions
  - Packet data (Header)
  - Packet metadata
  - Global and/or flow table metadata
- Action definitions
  - Invokable procedures that operate on packets and metadata
- Event definitions?
  - Are these variable or can we define a fixed set of event types?
    - Packet received
    - Timer expired
    - Condition met (e.g., threshold crossing)

What are we experimenting with?

- Parser
- Control Flow
- Hardware Abstraction

How is this done with OpenFlow datapath models today?

What can we add in the IR and what are the tradeoffs?

Parse Graph

OpenFlow TTP

- Parse graph implicit in OpenFlow
  - Or at least unspecified…
- Packet formats constrained by table pattern
  - Explicit match on protocol ID with fixed value
  - OpenFlow match field prerequisites
  - Can be further constrained by **`flow_paths`**

Parse Graph

air_meta.yaml

- **`parser`** provides a parse graph and protocol ID bindings
  - Parse graph is a directed graph connecting **`parse_state`** values
  - Binds protocol ID field values (**`select_value`**)
    - Also **`value_set`** can be referenced in parser
  - Enter parser and reach some parse state
    - Accepting and exception states (no exception feature yet)
  - Constrains packet structure
    - Prevents run-time binding of protocol ID values
      - E.g. Controller-defined Ethernet TPID
- Can the same information be derived from table definitions?
- Can the same constraints be provided in table definitions?
  - Depends on run-time flexibility in goto_table?
  - Use built-in table entries? Read-only tables?

Parse Graph

From discussion on 6 Feb 15 call :

- Also may need to indicate nesting level of a header
  - Each instance is explicitly defined by a separate **`header`** definition
  - May need more flexible scheme that allows reuse by not requiring a distinct header name per “level”
  - Three orthogonal concepts: field name, protocol ID value, location in packet
  - How is complexity of required parsing indicated in the IR?
  - How would MPLS entropy label be represented (two RFCs for this)?
- Tradeoff between
  - Requiring a distinct name for each field location in (any) packet
  - Allowing a field name to be (re)used in different contexts

Parse Graph

OF-PI

- OF-PI also allows run-time binding
  - E.g., as supported by POF and some hardware platforms

How can the IR support both
config-time binding and run-time binding?

MAT control

Mechanisms for composing actions

- Action list (**`APPLY_ACTIONS`** instruction)
  - Specified order of actions – config-time/run-time
- Action set (**`WRITE_ACTIONS`** instruction)
  - Canonical order of actions (datapath dependent) – spec-time
- Table match (MAT function)
  - Select highest priority matching table entry – config-time/run-time
- Table selection (**`GOTO_TABLE`** instruction)
  - Select next table from match context – config-time/run-time
- Control program
  - Imperative program guides MAT invocation – config-time

MAT control

OpenFlow TTP

- Table defines {<match, action>} pairs
- Constrains table entry types independent of pipeline location
- Pipeline order controlled by **`GOTO_TABLE`** instructions
  - But we want this to be more flexible for reusability

MAT control

air_meta.yaml (first version)

- `table` defines only match condition
  - Do we assume multi-feld match allows for arbitrary subsets?
    - Match types exact, ternary, valid
    - Separate logic from tables (e.g., all fields referenced by a table should be valid)
    - Or must this be explicit?
    - Two concerns
      - Specifying constraints explicitly
      - Transformation to an efficient implementation
  - If valid matches are A+B and A+C but not B+C, for example
    - Port+EtherType -> VNI (untagged ports)
    - Port+VID -> VNI (tagged ports)
    - Mapping complex function into a single table
      - Field A exact table, what if A is not valid? Various ambiguities. How to address these?
      - How exact must be IR be regarding run-time constraints on the datapath behavior?
    - Could use table matching on Port+VID+EtherType, but
      - Would this allow match on EtherType+VID? (invalid case)
      - Does preventing this require two tables?

MAT control

air_meta.yaml (current version)

- **`table`** defines match condition and valid actions
  - Does this provide sufficient constraint information?
    - Still relies on **`control_flow`** to provide additional constraints?

MAT control

air_meta.yaml (first version)

- `control_flow` defines match+action pair in pipeline context
  - Can a table be consulted multiple times in the pipeline?
    - Nothing prevents this currently; could add a constraint to prevent this
  - How does this constrain table entries?
    - E.g., what if a table entry carries a different action at run-time?
      - Run-time error checking (control context dependency)
  - Control flow decisions may depend on match context
    - Requires additional metadata

Hardware Abstraction

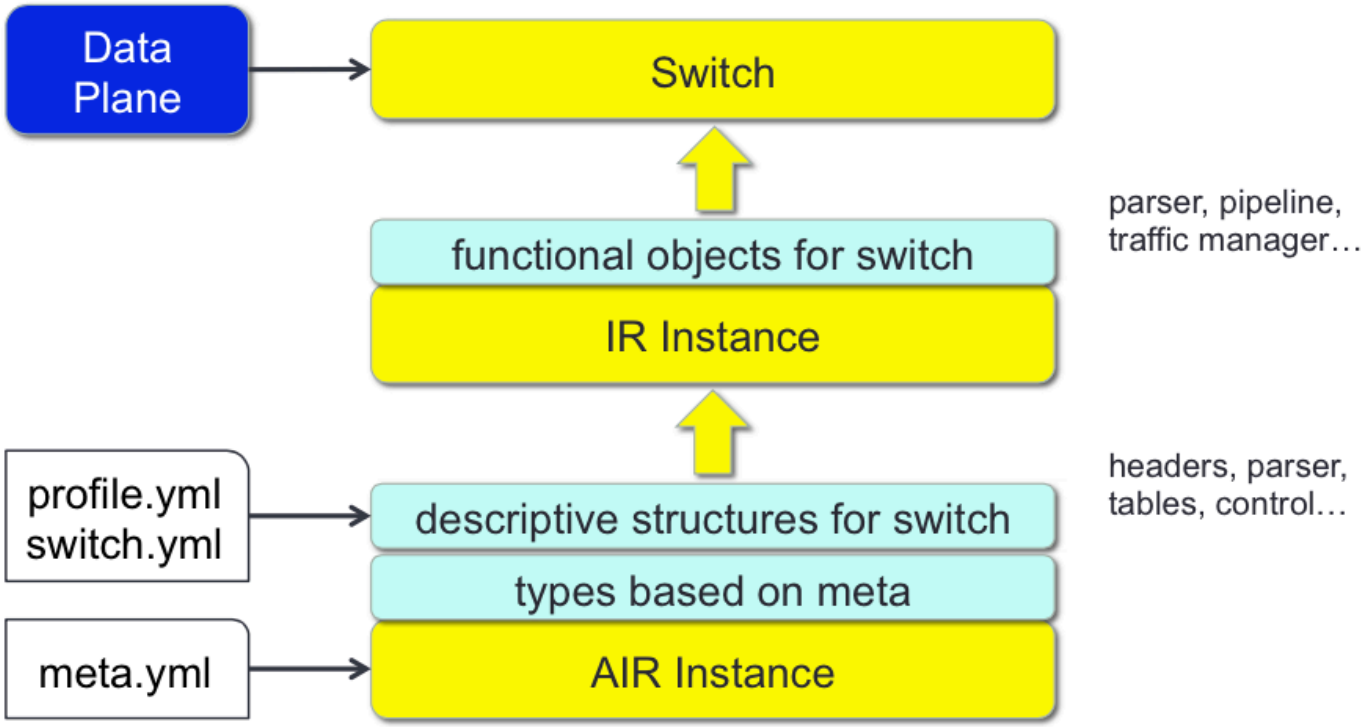
air_meta.yaml (first version)

- `processor_layout` defines a hardware model
- Is this fixed for a particular IR?
  - Or is this flexibility provided within the scope of the IR?
  - Either way is possible with the air_iri framework; processor_layout is expected to be changed less often
- What is the impact on building back-end (and front-end) compilers?
  - What about required vs. optional behaviors in the datapath model?
- Is a canonical hardware abstraction practical?
  - Not clear, we'll see as we try to used the layouts that are defined
  - Can check EXT JIRA to see what is being asked for there

Simulation and generated assets

IR Instance: Defines the behavior

AIR: Defines the domain



Defining the IR vs. Using the IR

- air_meta.yml (defines the IR components)
- action.py (defines primitives – needed before using?)

- ----- IR Specification -----
- headers
- actions (definitions, libraries)

- Tree of components with more essential components higher up
  - Children depend on parent
  - Draw a line where the desired IR functionality is above

FIN