AIR:
An IR for PIF

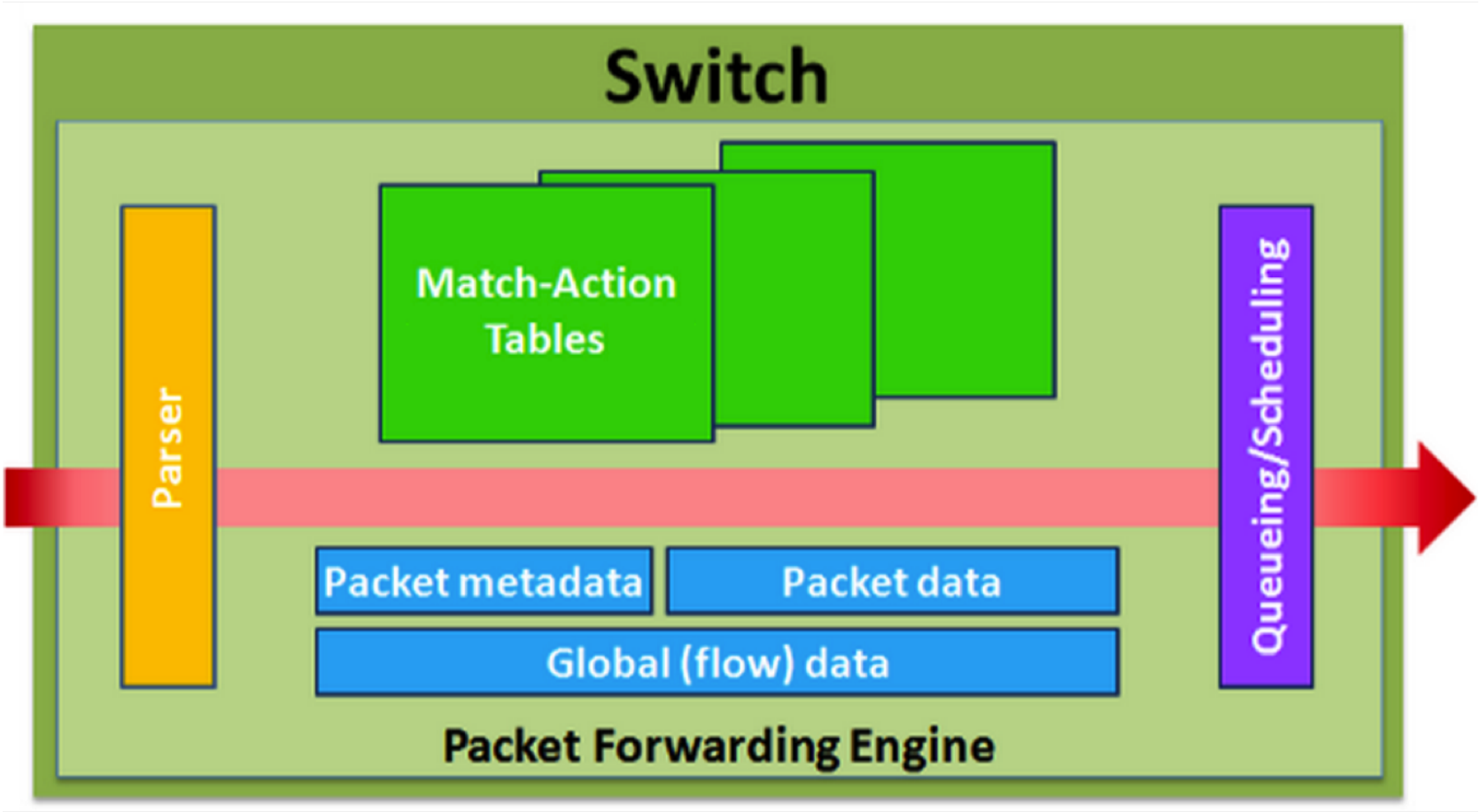
A proposed Intermediate Representation for Protocol Independent Forwarding

Dan Talayco
September, 2014

IR Overview

- Thin waist between
  - Multiple high level languages
  - Multiple target devices
- Driven by Abstract Forwarding Model
  - Should capture all semantics of that model
- Extensible and minimal

The Abstract Forwarding Model



What's in an Instance?

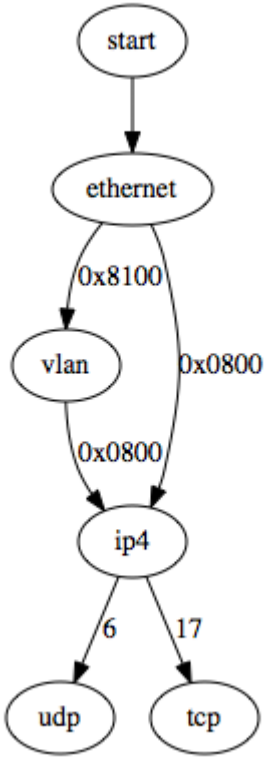
- Parse Graph
- Control Flow Graph
- Table Dependency Graph (maybe)
- Tables
- Fields and Headers
- Actions

Graphs are the dominant term

Hmm, what's good for graphs...

dot?

```
digraph G {
    start -> ethernet;
    ethernet -> vlan [label="0x8100"];
    ethernet -> ip4 [label="0x0800"];
    vlan -> ip4 [label="0x0800"];
    ip4 -> udp [label=6];
    ip4 -> tcp [label=17];
}
```



dot dot dot

- Many tools for dot
  - Visualization tools
  - Analysis and processing libraries import dot
- But…
  - We don't care about many dot attributes
  - We may want to add our own attributes
- So…
  - Assume we can update dot tools to support (or at least ignore) the attributes we want to add
  - More in a minute...

Syntax Proposal

- YAML for declarations and object attributes
  - But the metalanguage isn't that important
  - Can express these ideas in anything reasonable
- Define a scheme for legal IR instances
  - Build a validator as a first step
- Components
  - Tables, Fields, Actions, Parse Graph, CFG, TDG
- Graph representation is selectable
  - format : <supported-format>
  - Graph itself is a blob in the proper format
  - Will show with dot today

A Simple Example (excerpts)

Header declaration

```
ethernet :
  type : header
  doc : "The L2 header"
  fields :
    - dst_mac : 48
    - src_mac : 48
    - etherype : 16
```

Table declaration

```
l3 :
  type : table
  doc : "Do L3 intf selection"
  match_on :
    metadata.vfi : exact
    ipv4.dst : exact
```

Parser State

```
ethernet_p :
  type : parse_state
  extracts :
    - ethernet
  select_value :
    - ethernet.ethertype
```

Continued

Example, continued

Parse Graph

```
parser :
  type : parse_graph
  doc : "Implementation of primary parser"
  format : dot
  implementation : >
      start -> ethernet_p
      ethernet_p -> vlan_p [value=0x8100]
      ethernet_p -> ip4_p [value=0x0800]
      vlan_p -> ip4_p [value=0x0800]
      ip4_p -> udp_p [value=6]
      ip4_p -> tcp_p [value=17]
```

Continued

Example, continued

Control flow

```
ingress_flow :
  type : control_flow_graph
  format : dot
  # Use "action" here; really just a label
  implementation : >
    vlan -> l2 [action="set_l2_vfi_a"]
    vlan -> l3 [action="set_l3_vfi_a"]
    l2 -> flood_block [action="flood"]
    l2 -> queue [action="set_egress_spec"]
    l3 -> apply_route [action="set_route"]
    apply_route -> queue [action="apply_route_if"]
```

Also part of the IR

- Fields
  - Identify and define attributes
  - Support extended attributes
- Actions
  - The list of primitive actions
    - An explanation of their semantics
    - Mechanism to extend the list…but shouldn't encourage
  - Define semantics for combining primitive actions into actions used by tables
- Table matching semantics
  - Define: exact, ternary, etc

Details of AIR

- Instance driven declaration
- Each AIR object has explicit type specified
- Object types shown on next slide
  - field_type, header, etc
- All AIR objects must have 'type' and 'doc' attributes
- May use YAML mechanisms such as references

AIR object types and their supported attributes

- field_type
  - width
  - signed
  - saturating
  - ...
- header
  - fields (list)
- parser_state
  - extracts
  - select_value
- parse_graph
  - format
  - implementation

Continued

AIR object types, continued

- action
  - format
  - parameter_list
    - each w/ attrs
  - implementation
- table
  - match_on
- control_flow_graph
  - format
  - implementation
- dependency_graph
  - format
  - implementation

Proposed Timeline

- Validator for AIR
  - Q4 2014
- Simulator for AIR
  - Q1 2015
- AIR tools
  - Visualizer, editors, formal model checking
  - Starting in Q4 2014
- AIR for OVS
  - Q1/Q2 2015

Questions

- Binding Time
  - Should AIR initially support run-time binding of field names for action primitives?
  - Proposal: No.
    - Reduces (removes?) need for TDG
    - Consider support in the future
- Options for Semantics for Action Primitives
  - Documentation
    - Optionally with reference implementation
  - Define underlying compute model and give algos

Tasks and Sign-up Sheet

- Details for AIR-YAML specification
- Complete "basic functionality" example
- Simulations
  - Python
  - HW Simulation Environment driven
  - (more) performant C-based model
- OVS Integration
- Visualization Tools (packet tracker?)
- Devise Debugging and Monitoring tools
- Interactive Designer