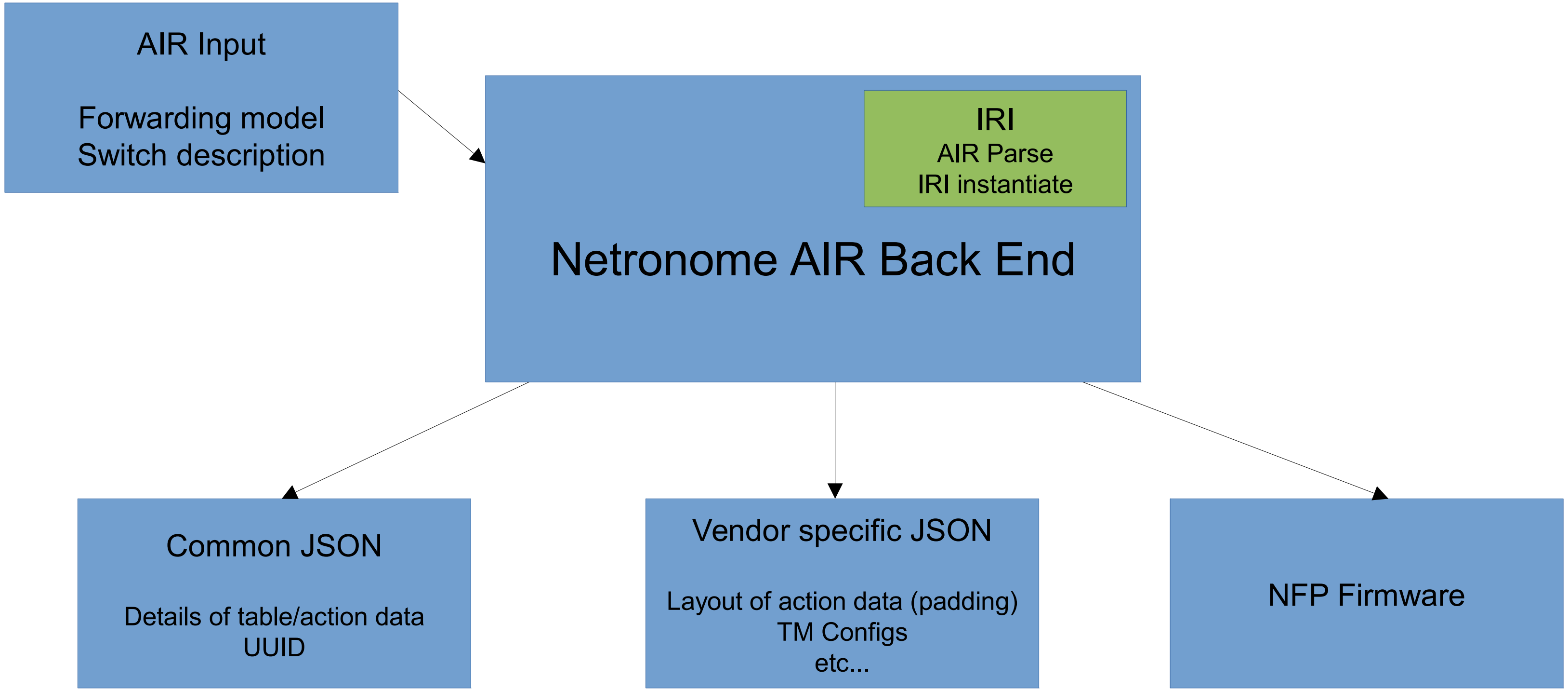
PIF Runtime Discussion

14 April 2015

Overview

- Will briefly:
  - Look at prototype NFP PIF back end
  - Discuss NFP prototype run time interfaces
  - Propose low level interface solution
    - Won't cover API specifics

Netronome PIF back end prototype



Prototype runtime interface

- Loading / Unloading
  - Shell scripts using platform config + elf as input
- Table configuration
  - Simple C tool
  - Common + vendor JSON to configure NFP and perform sanity check on rule input
  - rules.json file containing match fields + action data for tables
- Statistics
  - Shell scripts

Standardizing on interfaces

- Common interface will be a great benefit
  - low duplication of effort
  - users environment familiar; tool reuse etc
- Likely interfaces to cover:
  - Table interaction: discovery, status and configuration
  - Statistics, port management, loading/unloading?
- Standardization on intermediate back end outputs?

Proposed implementation

- Apache thrift for interface definition + RPC
  - thrift.apache.org/
  - Easily integrates with many languages (c++/python)
  - Supports nifty types: maps, sets, lists
  - Fast
- RPC functions and data structures defined in .swift file
  - This would be first port-of-call: define interfaces and data structures for PIF designs
- Switch/vendor specific code in server compiled with RPC stubs; client neutral

Server/client example

Python Server

```
class PifInterface:
  def __init__(self):
    self.log = {}

  def ping(self):
    print 'ping()'

  def get_table_info():
    return pif_table_info
```

C++ Client Snippet

```
TTransport socket(new TSocket("localhost",
9090));
TTransport transport(new
TBufferedTransport(socket));
TProtocol protocol(new
TBinaryProtocol(transport));
PifInterface client(protocol);
TableInfo table_info;

transport->open();

client.ping();

table_info = client.get_table_info();
printf(“PIF design has %d tables\n”,
table_info.cnt);
```

Thoughts?

...