Software Defined Networking

Dr. Nick Feamster
Professor

In this course, you will learn about software defined networking and how it is changing the way communications networks are managed, maintained, and secured.

This Lesson: Protocol Independent Packet Processing

- Motivation
- Two examples
  - P4: Programming Protocol-Independent Packet Processors (main focus)
  - POF: Protocol Oblivious Forwarding

Material adapted from Jennifer Rexford's talk at ONS 2014

Over the Past Five Years...

Version	Date	# Headers
OF 1.0	Dec 2009	12
OF 1.1	Feb 2011	15
OF 1.2	Dec 2011	36
OF 1.3	Jun 2012	40
OF 1.4	Oct 2013	41

- Control and data not sufficiently decoupled
- No easy way to modify packet format
- Adding new features requires changing FE and controller

Desirable Features in SDN Switches

- Configurable packet parser
  - Not tied to a specific header format
- Flexible match+action tables
  - Multiple tables (in series and/or parallel)
  - Able to match on all defined fields
- General packet-processing primitives
  - Copy, add, remove, and modify
  - For both header fields and meta-data

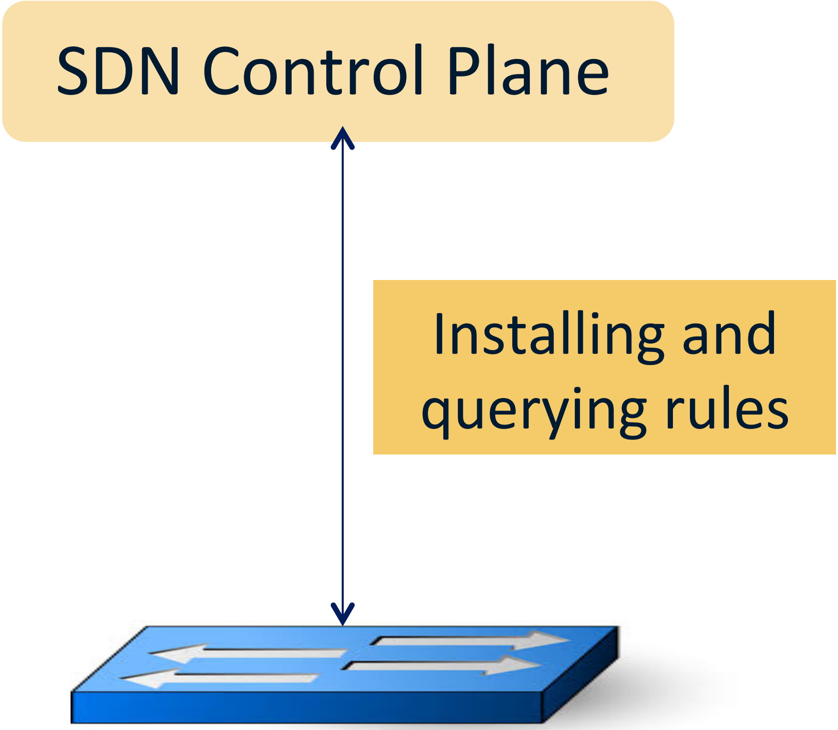
New Hardware Makes This Possible

- New generation of switch ASICs
  - Intel FlexPipe
  - RMT [SIGCOMM'13]
  - Cisco Doppler
- But, programming these chips is hard
  - Custom, vendor-specific interfaces
  - Low-level, akin to microcode programming

Three Goals

- Protocol independence
  - Configure a packet parser
  - Define a set of typed match+action tables
- Target independence
  - Program without knowledge of switch details
  - Rely on compiler to configure the target switch
- Reconfigurability
  - Change parsing and processing in the field

"Classic" OpenFlow (1.x)

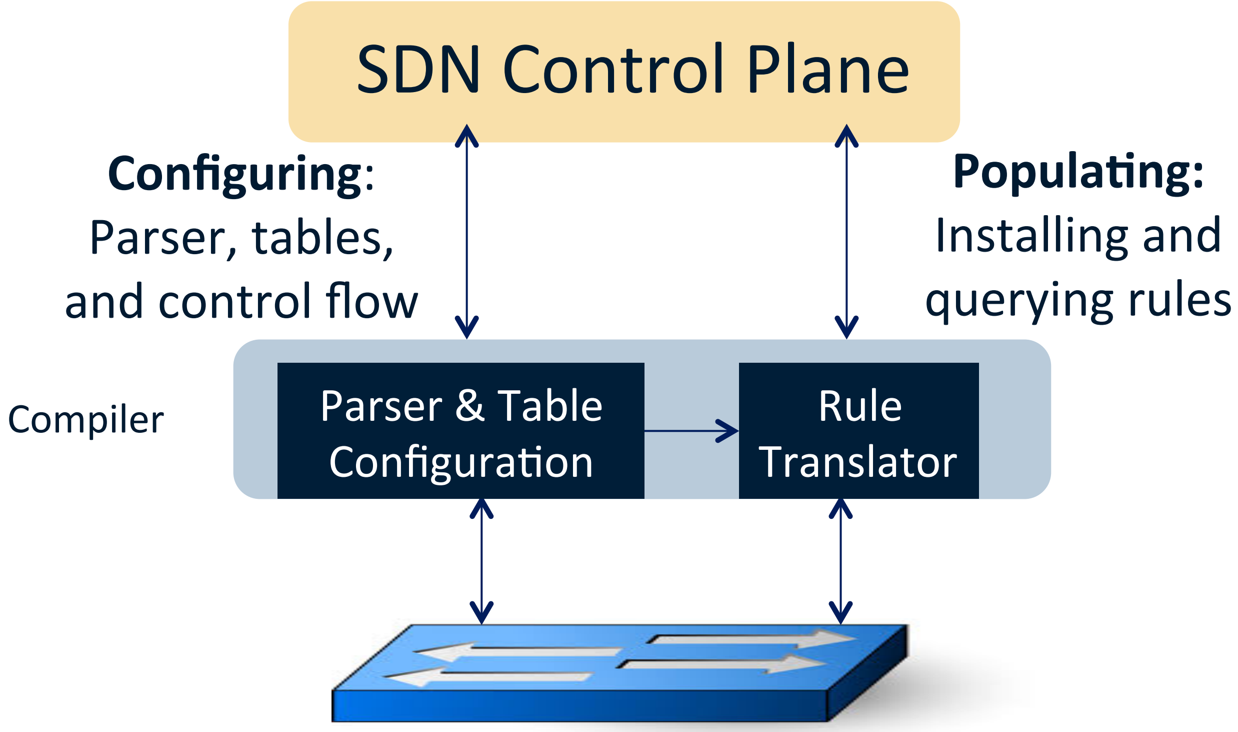


SDN Control Plane

Installing and querying rules

Software Defined Networking

"OpenFlow 2.0"



SDN Control Plane

Configuring:
Parser, tables,
and control flow

Populating:
Installing and
querying rules

Compiler

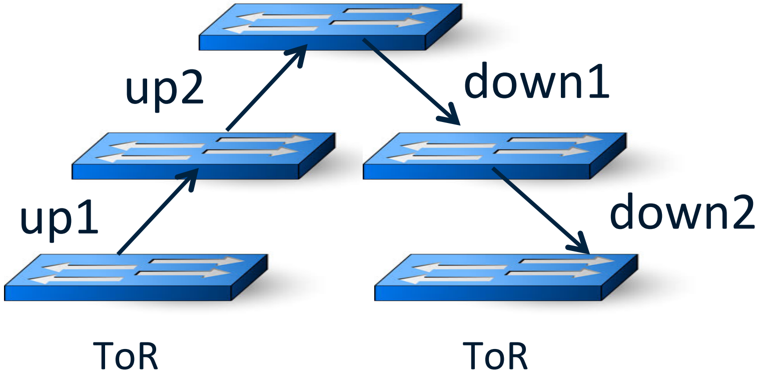
Parser & Table
Configuration

Rule
Translator

Simple Motivating Example

- Data-center routing
  - Top-of-rack switches
  - Two tiers of core switches
  - Source routing by ToR

- Hierarchical tag (mTag)
  - Pushed by the ToR
  - Four one-byte fields
  - Two hops up, two down



Header Formats

- Ordered list of fields
- A field has a name and width

```
header ethernet {
  fields {
    dst_addr : 48;
    src_addr : 48;
    ethertype : 16;
  }
}
```

```
header vlan {
  fields {
    pcp : 3;
    cfi : 1;
    vid : 12;
    ethertype : 16;
  }
}
```

```
header mTag {
  fields {
    up1 : 8;
    up2 : 8;
    down1 : 8;
    down2 : 8;
    ethertype : 16;
  }
}
```

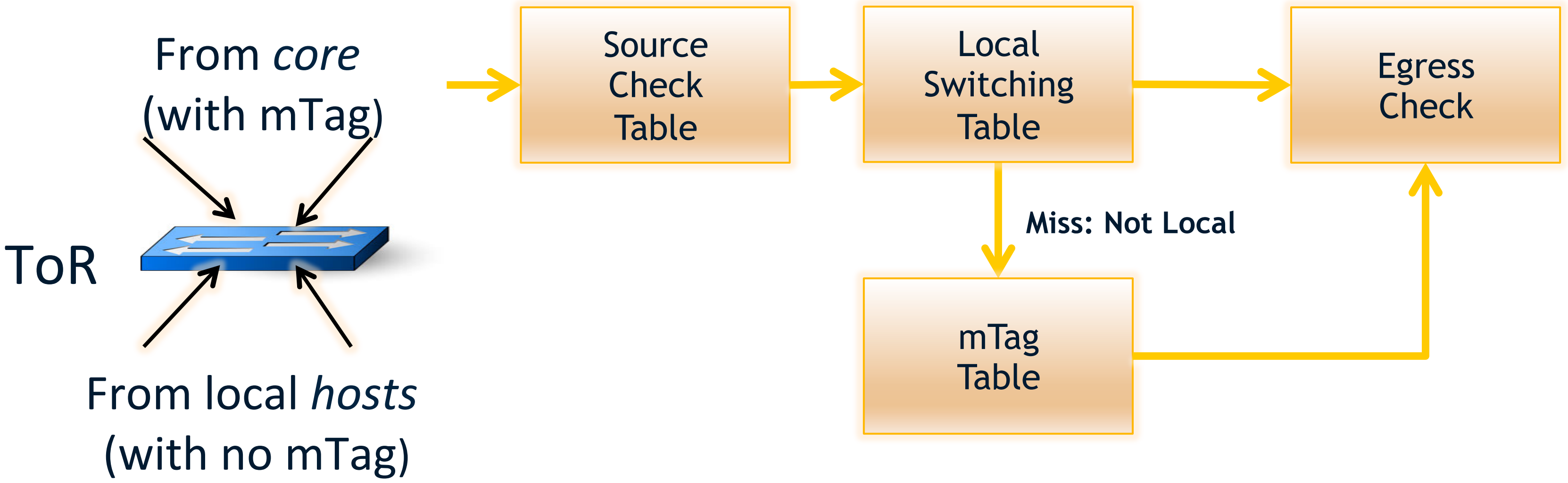
Typed Tables

- Describe each packet-processing stage
  - What fields are matched, and in what way
  - What action functions are performed
  - (Optionally) a hint about max number of rules

```
table mTag_table {
  reads {
    ethernet.dst_addr : exact;
    vlan.vid : exact;
  }
  actions {
    add_mTag;
  }
  max_size : 20000;
}
```

Control Flow

- Flow of control from one table to the next
  - Collection of functions, conditionals, and tables



P4 Compiler

- Parser
  - **Programmable parser:** translate to state machine
  - **Fixed parser:** verify the description is consistent
- Control program
  - **Target-independent:** table graph of dependencies
  - **Target-dependent:** mapping to switch resources
- Rule translation
  - Verify that rules agree with the (logical) table types
  - Translate rules to tables

Compiling to Target Switches

- Software switches
  - Directly map the table graph to switch tables
  - Use data structure for exact/prefix/ternary match
- Hardware switches with RAM and TCAM
  - RAM: hash table for tables with exact match
  - TCAM: for tables with wildcards in the match
- Switches with parallel tables
  - Analyze table graph for possible concurrency

Compiling to Target Switches

- Applying actions at the end of pipeline
  - Instantiate tables that generate meta-data
  - Use meta-data to perform actions at the end
- Switches with a few physical tables
  - Map multiple logical tables to one physical table
  - “Compose” rules from the multiple logical tables
  - … into “cross product” of rules in physical table

Conclusion

- OpenFlow 1.x
  - Vendor-agnostic API
  - But, only for fixed-function switches
- An alternate future
  - Protocol independence
  - Target independence
  - Reconfigurability in the field
- P4: a strawman proposal
  - Other proposals: POF