Sharif University of Technology

Computer Engineering Department

Software-Defined Networking

Ali Movaghar

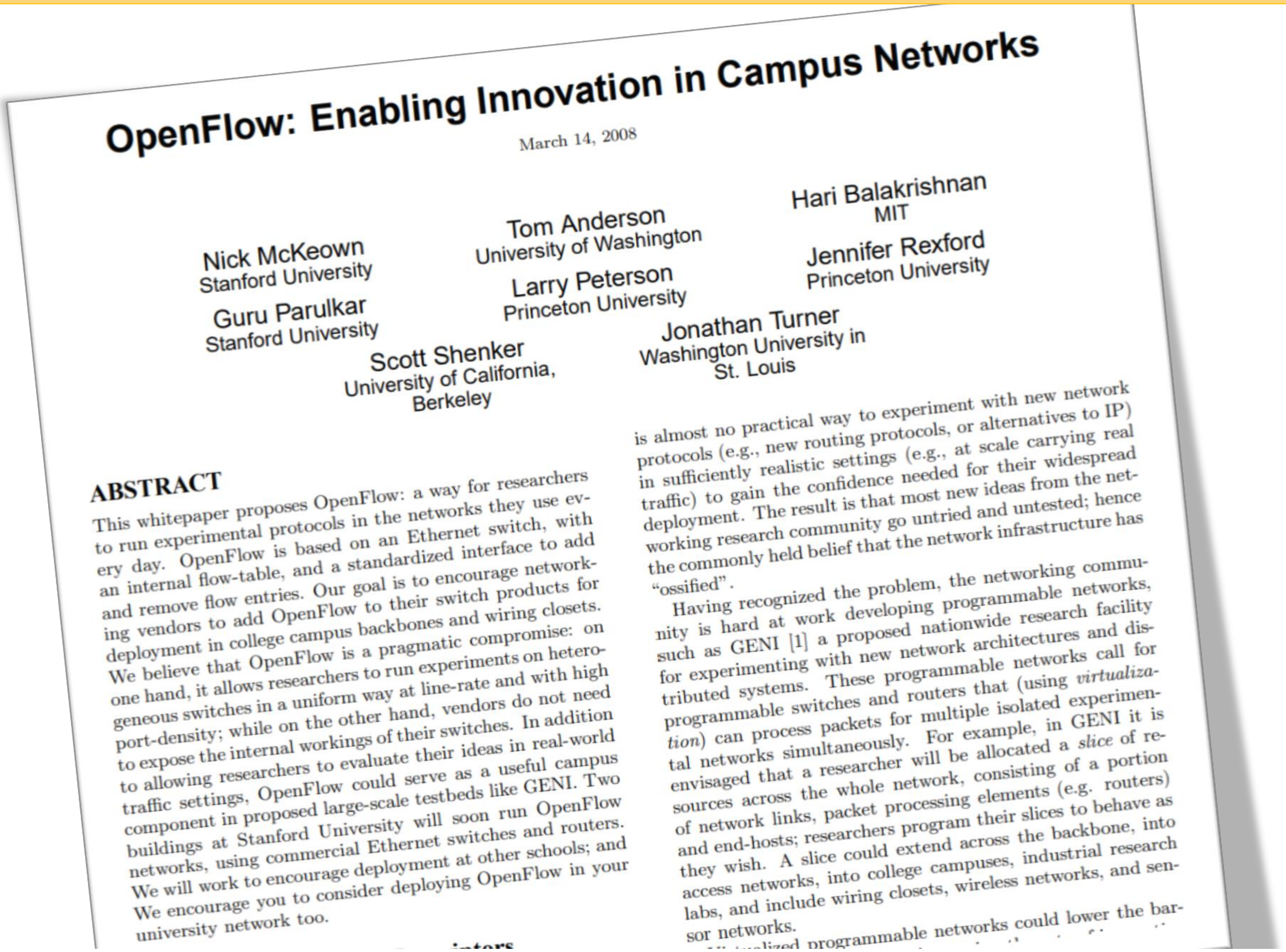
Mohammad Hosseini

OpenFlow – Part 1

TA: Iman Rahmati & Farbod Shahinfar

OpenFlow paper

ACM SIGCOMM computer communication review 38.2 (2008)



OpenFlow: Enabling Innovation in Campus Networks

March 14, 2008

Nick McKeown
Stanford University

Tom Anderson
University of Washington

Hari Balakrishnan
MIT

Guru Parulkar
Stanford University

Larry Peterson
Princeton University

Jennifer Rexford
Princeton University

Scott Shenker
University of California, Berkeley

Jonathan Turner
Washington University in St. Louis

ABSTRACT

This whitepaper proposes OpenFlow: a way for researchers to run experimental protocols in the networks they use every day. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal is to encourage networking vendors to add OpenFlow to their switch products for deployment in college campus backbones and wiring closets. We believe that OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI. Two buildings at Stanford University will soon run OpenFlow networks, using commercial Ethernet switches and routers. We will work to encourage deployment at other schools; and We encourage you to consider deploying OpenFlow in your university network too.

is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment. The result is that most new ideas from the networking research community go untried and untested; hence the commonly held belief that the network infrastructure has "ossified".

Having recognized the problem, the networking community is hard at work developing programmable networks, such as GENI [1] a proposed nationwide research facility for experimenting with new network architectures and distributed systems. These programmable networks call for programmable switches and routers that (using *virtualization*) can process packets for multiple isolated experimental networks simultaneously. For example, in GENI it is envisaged that a researcher will be allocated a *slice* of resources across the whole network, consisting of a portion of network links, packet processing elements (e.g. routers) and end-hosts; researchers program their slices to behave as they wish. A slice could extend across the backbone, into access networks, into college campuses, industrial research labs, and include wiring closets, wireless networks, and sensor networks.

The need for programmable networks

- The reduction in real-world impact of any given network innovation
- No practical way to experiment with new network protocols in realistic settings to gain the confidence needed for their widespread deployment
- Most new ideas from the networking research community go untried and untested
- The network infrastructure has "ossified"

As researchers,
how can we run experiments in our campus networks?

A naive approach

- Persuading commercial “name-brand” equipment vendors to provide an open, programmable, virtualized platform on their switches and routers so that researchers can deploy new protocols.
- Very unlikely
- The practice of commercial networking is that the standardized external interfaces are narrow, and all of the switch’s internal flexibility is hidden.
- Network equipment vendors have spent years deploying and tuning fragile distributed protocols and algorithms, and they fear that new experiments will bring networks crashing down.
- Open platforms lower the barrier-to-entry for new competitors.

The approach of using open platforms

- ❖ A PC with several network interfaces
- ❖ Open-source implementations of routing protocols exist (e.g., Click)
- ❖ This solution does not have the performance or port-density that is needed

A promising approach

- Compromise on generality and to seek a degree of switch flexibility that is:
  - Amenable to high-performance and low-cost implementations
  - Capable of supporting a broad range of research
  - Assured to isolate experimental traffic from production traffic
  - Consistent with vendors' need for closed platforms

The OpenFlow Switch

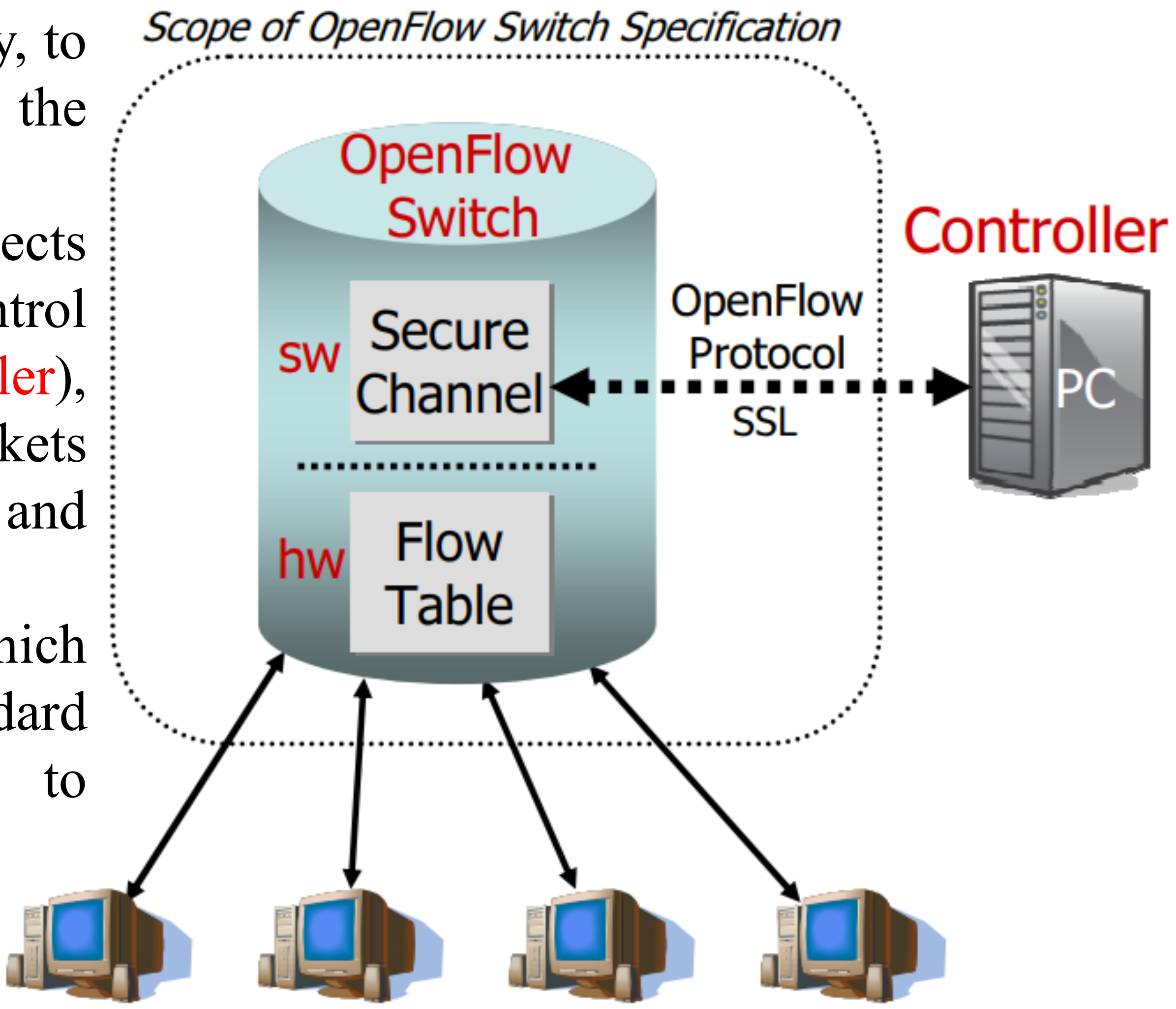
The OpenFlow Switch

- ❖ Exploiting the fact that most modern Ethernet switches and routers contain flow-tables (typically built from TCAMs) that run at line-rate.
- ❖ OpenFlow exploits a common set of functions that run in many switches and routers.
- ❖ OpenFlow provides an open protocol to program the flow-table in different switches and routers.

The OpenFlow Switch

An OpenFlow Switch consists of at least three parts:

- ❖ A Flow Table, with an action associated with each flow entry, to tell the switch how to process the flow
- ❖ A Secure Channel that connects the switch to a remote control process (called the controller), allowing commands and packets to be sent between a controller and the switch using
- ❖ The OpenFlow Protocol, which provides an open and standard way for a controller to communicate with a switch



The OpenFlow Switch categories

- Dedicated OpenFlow switches
  - Do not support normal Layer 2 and Layer 3 processing
  - Dumb datapath elements that forward packets between ports, as defined by a remote controller

- OpenFlow-enabled switches
  - Commercial switches and routers to which the OpenFlow protocol and interfaces have been added as a new feature

Actions

Each flow-entry has a simple action associated with it.

1. Forward this flow’s packets to a given port (or ports)
2. Encapsulate and forward this flow’s packets to a controller
   - Typically used for the first packet in a new flow
3. Drop this flow’s packets
4. Forward this flow’s packets through the switch’s normal processing pipeline (applicable to OpenFlow-enabled switches)

Flow-Table Entries

Header (match)	Action	Statistics

An entry in the Flow-Table has three fields:

1. A packet header that defines the flow

In	VLAN	Ethernet			IP			TCP	
Port	ID	SA	DA	Type	SA	DA	Proto	Src	Dst

➢ Each header field can be a wildcard to allow for aggregation of flows

2. The action, which defines how the packets should be processed

3. Statistics, which keep track of the number of packets and bytes for each flow, and the time since the last packet matched the flow (to help with the removal of inactive flows)

Switch types

- If a switch supports the header formats and the four basic actions mentioned above then we call it a "Type 0" switch.

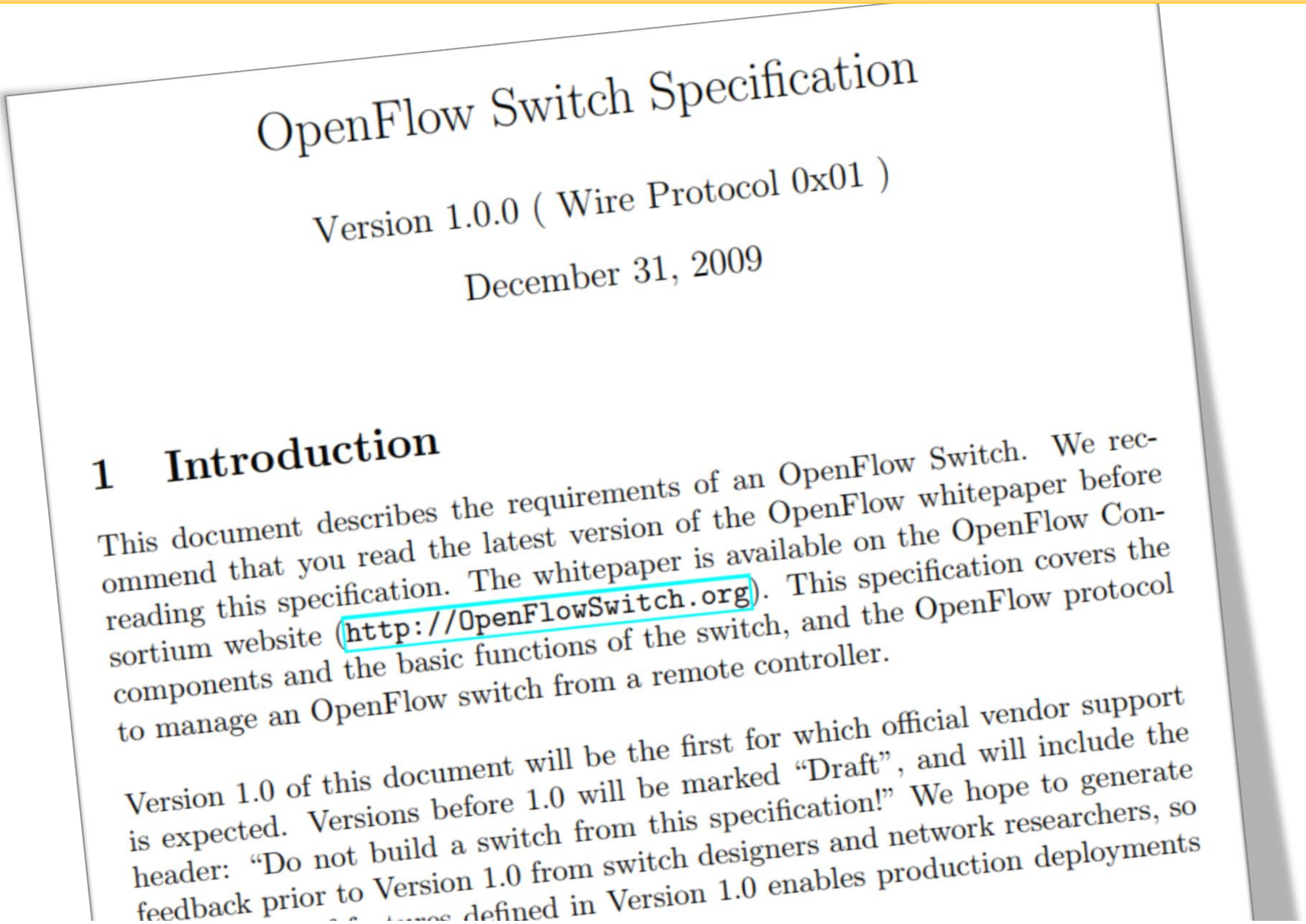
- It is expected that many switches will support additional actions, for example to rewrite portions of the packet header (e.g., for NAT).
- As a particular set of features emerges, we will define a "Type 1" switch.

In practice, next versions of OpenFlow were released without mentioning the term "switch type".

OpenFlow Consortium and OpenFlow Switch Specification

- ❖ The paper refers readers to *OpenFlow Switch Specification* for detailed requirements and definitions. (but the document was a draft version, and version 1.0 was released in 2009 with some differences compared to the paper)
- ❖ The standard is maintained by *OpenFlow Consortium*
- ❖ The consortium is a group of researchers and network administrators aimed at popularizing OpenFlow.
- ❖ In 2011, the consortium was renamed Open Networking Foundation

OpenFlow Switch Specification Version 1.0.0 - 2009



OpenFlow Switch Specification

Version 1.0.0 (Wire Protocol 0x01)

December 31, 2009

1 Introduction

This document describes the requirements of an OpenFlow Switch. We recommend that you read the latest version of the OpenFlow whitepaper before reading this specification. The whitepaper is available on the OpenFlow Consortium website (http://OpenFlowSwitch.org). This specification covers the components and the basic functions of the switch, and the OpenFlow protocol to manage an OpenFlow switch from a remote controller.

Version 1.0 of this document will be the first for which official vendor support is expected. Versions before 1.0 will be marked "Draft", and will include the header: "Do not build a switch from this specification!" We hope to generate feedback prior to Version 1.0 from switch designers and network researchers, so

defined in Version 1.0 enables production deployments

Header fields for matching

- 12 fields from packets were used to match against flow entries

Ingress port			
Ethernet source	Ethernet destination	Ethernet type	
VLAN ID	VLAN priority		
IP source	IP destination	IP protocol	IP ToS
TCP/UDP source port	TCP/UDP destination port		

Counters

Per Table	Bits
Active Entries	32
Packet Lookups	64
Packet matches	64

Per Port	Bits
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64
Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
… Errors	

Per Flow	Bits
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32

Per Queue	Bits
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

Actions

Required Actions

- **Forward**
  - To a physical ports
  - To a virtual port
    - ALL, CONTROLLER, LOCAL, TABLE, IN_PORT
- **Drop**

Optional Actions

- **Forward**
  - NORMAL
  - FLOOD

- **Enqueue**
- **Modify-Field**

Message Types

- ❖ **Controller-to-Switch**
  - ➢ Initiated by the controller and used to directly manage or inspect the state of the switch
- ❖ **Asynchronous**
  - ➢ Initiated by the switch and used to update the controller of network events and changes to the switch state
- ❖ **Symmetric**
  - ➢ initiated by either the switch or the controller and sent without solicitation

Controller-to-Switch Messages

Controller-to-Switch messages are initiated by the controller and may or may not require a response from the switch.

- ❖ **Features (FEATURES_REQUEST – FEATURES_REPLY)**
  - ➢ Upon TLS session establishment (TCP port: 6633), the controller sends a features request message to the switch. The switch must reply with a features reply that specifies the capabilities supported by the switch
- ❖ **Configuration (SET_CONFIG – GET_CONFIG_REQUEST – GET_CONFIG_REPLY)**
  - ➢ To set and query configuration parameters in the switch
- ❖ **Modify-State (FLOW_MOD – PORT_MOD)**
  - ➢ To manage states on the switches.
  - ➢ To add/delete and modify flows in the flow tables and to set switch port properties.

Controller-to-Switch Messages (Cont.)

- **Read-State (STATS_REQUEST – STATS_REPLY)**
  - To collect statistics from the switch’s flow-tables, ports and the individual flow entries
- **Send-Packet (PACKET_OUT)**
  - Send packets out of a specified port on the switch.
- **Barrier (BARRIER_REQUEST – BARRIER_REPLY)**
  - To ensure message dependencies have been met or to receive notifications for completed operations

Asynchronous Messages

Switches send asynchronous messages to the controller to denote a packet arrival, switch state change, or error.

- **Packet-in (PACKET_IN)**
  - For all packets that do not have a matching flow entry, or if a packet matches an entry with a "send to controller" action.
  - Contains a buffer ID.
  - Switches that do not support internal buffering (or have run out of internal buffering) must send the full packet to the controller as part of the event.
- **Flow-Removed (FLOW_REMOVED)**
  - Notify the controller of a flow removal due to *idle_timeout* or *hard_timeout*.
- **Port-Status (PORT_STATUS)**
- **Error (FLOW_MOD – PORT_MOD)**
  - Notify the controller of problems.

Symmetric Messages

Symmetric messages are sent without solicitation, in either direction.

- **Hello** **(HELLO)**
  - When an OpenFlow connection is first established, each side of the connection must immediately send a HELLO message with the version field set to the highest OpenFlow protocol version supported by the sender.
  - The smaller version number is selected as the common version to be used.
- **Echo** **(ECHO_REQUEST – ECHO_REPLY)**
  - To determine the latency, bandwidth, and/or liveness of a controller-switch connection.