

COPYRIGHTED BY

Satyajeet Padmanabhi

May 2015

CREATION OF FLEXIBLE DATA STRUCTURE FOR AN EMERGING NETWORK CONTROL PROTOCOL

A Thesis

Presented to

The Faculty of the Department of Engineering Technology

University of Houston

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science in Engineering Technology

By

Satyajeet Padmanabhi

May 2015

**CREATION OF FLEXIBLE DATA STRUCTURE FOR AN
EMERGING NETWORK CONTROL PROTOCOL**

Satyajeet Padmanabhi

Approved:

Chair of the Committee
Deniz Gurkan, Ph.D.
Associate Professor
Engineering Technology

Committee Members:

Ricardo Lent, Ph.D.
Assistant Professor
Engineering Technology

Mequanint Moges, Ph.D.
Assistant Chair
Engineering Technology

Rupa Iyer, Ph.D.
Associate Dean for Research and
Graduate Studies
College of Technology

Wajiha Shireen, Ph.D.
Department Chair
Engineering Technology
College of Technology

Acknowledgments

I would like to express my deep and sincere gratitude to my advisor, Dr. Deniz Gurkan. Her invaluable advice and guidance have provided a good basis for this thesis. I am very thankful for her ideas, comments and consistent support throughout this work. My special thanks to committee members Dr. Ricardo Lent and Dr. Mequanint Moges for their valuable suggestions and help. I would like to thank Sandhya Narayan and Dmitry Orekhov from Infoblox for their support during my work at University of Houston. I am also thankful to my parents and all friends for all their support and motivation.

Abstract

Due to increasing number of versions of OpenFlow protocol, it is getting harder day by day to use isolated data structure support of OpenFlow protocol. There is high degree of variability between each versions of OpenFlow protocol. Each version of OpenFlow specifies an interface and the collection of abstractions present in a switch that can be manipulated. So our focus of this thesis is to use the data structure (Avro) which supports OpenFlow protocol through software infrastructure proposed by Warp development group. Using this we have developed the OpenFlow version 1.2 support to Warp controller. Warp architecture uses Avro data structure which has advantages like easy integration of new version, update existing version and apply run time changes, version control, data exchange and easy schema processing which heavily impact on performance and flexibility of OpenFlow controller. These mentioned factors are compared against other OpenFlow controller architectures such as Floodlight, Ryu etc. Comparing obtained observations with different architectures conclude that Warp is more flexible architecture as compared to Floodlight and Ryu.

Table of Contents

Acknowledgments	4
Abstract	5
Chapter 1 – Introduction	10
1.1 Software Defined Networking (SDN)	10
1.2 OpenFlow Protocol	12
1.3 SDN Controllers	13
1.4 Thesis Work	14
1.5 Literature survey	15
1.6 Thesis Overview	16
Chapter 2 – Data Structure (Avro) and Warp Architecture	18
2.1 Avro	18
2.1.1 Schemas	20
2.1.2 Comparison with other systems	21
2.2 Warp Architecture with inclusion of OpenFlow version 1.2	22
Chapter 3 – OpenFlow	26
3.1 OpenFlow Version 1.0	29
3.1.1 Slicing	29
3.1.2 Flow Cookies.....	29
3.1.3 User-specifiable data path description	30
3.1.4 Match on IP fields in ARP packets	30
3.1.5 Match on IP ToS/DSCP bits.....	30
3.1.6 Querying port stats for individual ports.....	30
3.1.7 Improved flow duration resolution in stats/expiry messages	30
3.1.8 Other changes to the Specification	31
3.2 OpenFlow Version 1.1	31
3.2.1 Multiple Tables.....	31
3.2.2 Groups	34

3.2.3 MPLS & VLAN	34
3.2.4 Virtual Ports.....	35
3.2.5 Controller connection failure	35
3.2.6 Other changes	36
3.3 OpenFlow Version 1.2	36
3.3.1 Extensible Match Support	37
3.3.2 Extensible 'set field' packet rewriting support	37
3.3.3 Extensible context expression in 'packet-in'	37
3.3.4 Extensible Error messages via experimenter error type.....	38
3.3.5 IPv6 support added	38
3.3.6 Simplified behavior of flow-mod request	38
3.3.7 Removed packet parsing specification.....	39
3.4 OpenFlow Version 1.3	39
3.4.1 Refactor capabilities negotiation	39
3.4.2 More flexible table miss support	40
3.4.3 IPv6 Extension Header handling support.....	41
3.4.4 Per Flow meters	42
3.4.5 Per connection event filtering.....	42
3.4.6 Auxiliary connections	43
3.4.7 MPLS BoS matching.....	44
3.4.8 Provider Backbone Bridging tagging	44
3.4.9 Rework tag order.....	44
3.4.10 Tunnel-ID metadata	45
3.4.11 Cookies in packet-in	45
3.4.12 Duration for stats	46
3.4.13 On-demand flow counters	46
3.4.14 Other changes	46
3.5 OpenFlow Version 1.4	47
3.5.1 More extensible wire protocol.....	47
3.5.2 More descriptive reasons for packet-in	48

3.5.3 Optical port properties.....	49
3.5.4 Flow- removed reason for meter delete.....	49
3.5.5 Flow monitoring.....	50
3.5.6 Role status event.....	51
3.5.7 Eviction.....	51
3.5.8 Vacancy events.....	52
3.5.9 Bundles.....	53
3.5.10 Synchronized tables.....	53
3.5.11 Group and meter change notifications.....	54
3.5.12 Error code and bad priority.....	54
3.5.13 Error code and set-async-config.....	54
3.5.14 PBB UCA header field.....	55
3.5.15 Error code for duplicate instruction.....	55
3.5.16 Error code for multipart timeout.....	56
3.5.17 Change default TCP port to 6653.....	56
Chapter 4 – Warp comparison with other OpenFlow Controllers.....	57
4.1 Integrate new version of OpenFlow.....	57
4.2 Update existing version/Run time changes.....	58
4.3 Version control.....	58
4.4 Data exchange.....	59
4.5 Code processing.....	59
4.6 Architecture Comparison.....	61
Chapter 5 – Results, Conclusion and Future Work.....	63
References.....	64
Appendix: A – Code.....	67
Appendix: B – Credits.....	106

List of figures

Figure 1: SDN Architecture	11
Figure 2: Data Definition (Avro)	21
Figure 3: Data exchange between different languages	22
Figure 4: Warp architecture	23
Figure 5: Actors of Warp controller.....	25
Figure 6: OpenFlow overview	26
Figure 7: Protocol decomposition.....	27
Figure 8: Message Layer.....	29
Figure 9: (a) Compilation process of source code (b) Processing of schema.....	60

List of tables

Table 1: Comparison of OpenFlow architectures	61
Table 2: Flexibility (%).....	62

Chapter 1 – Introduction

Due to increasing number of versions of OpenFlow protocol, it is getting harder day by day to use isolated data structure support of OpenFlow protocol. There is high degree of variability between each versions of OpenFlow protocol. Each version of OpenFlow specifies an interface and the collection of abstractions present in a switch that can be manipulated. So our focus of this thesis is to use the data structure (Avro) which supports OpenFlow protocol through software infrastructure proposed by Warp development group. Using this we have developed the OpenFlow version 1.2 support to Warp controller.

1.1 Software Defined Networking (SDN)

Traditional networks are complex and hard to manage due to two reasons: one is that the control plane and data planes are vertically integrated and vendor specific, another concurring reason is that typical networking devices are also tightly tied to line products and versions. Software defined networking (SDN) is created to overcome these problems [14]. SDN, often referred to a “radical new idea in networking”, promises to dramatically simplify network management and enable innovation through network programmability [10, 24] and could simplify network operations and reduces cost of the network [11]. With SDN, network management moves from codifying functionality in terms of low-level device configurations to building software that facilitates network management and debugging [12]. SDN is an emerging network architecture in which network’s control plane and forwarding

plane gets separated physically. This architecture is manageable, dynamic, cost effective and adaptable for today's network requirements. As it separates network control and forwarding functionalities, it results network control directly programmable. This approach is possible with the help of OpenFlow protocol. Control plane is responsible for making decisions regarding traffic of the network. It's responsible for deciding where traffic should go and also for system configuration, management and exchange of routing table information. Forwarding plane, also known as data plane, forwards network traffic to the next hop along the path which is decided by control plane. SDN gives network administrator better control over the network thus increasing network performance [9].

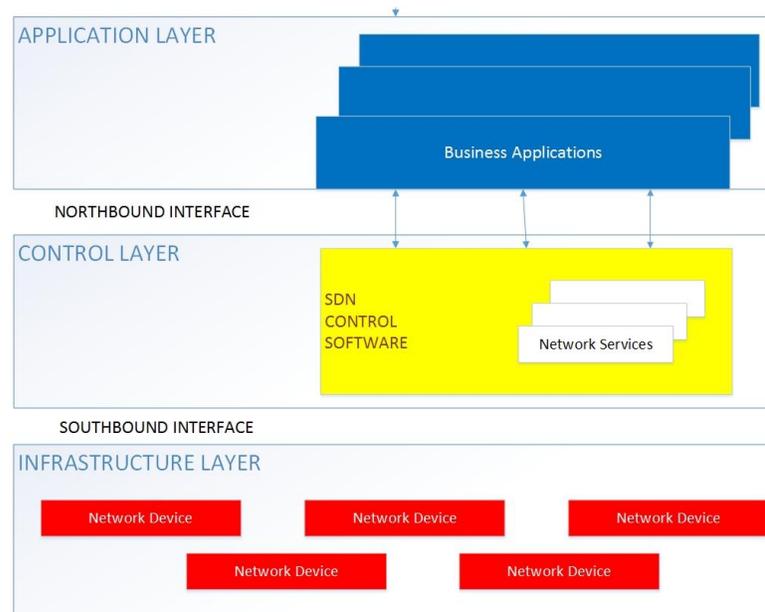


Figure 1: SDN Architecture

Figure 1 shows SDN architecture which consists of 3 layers: application layer, control layer and infrastructure layer. Application layer consist of business applications and it communicates with control layer using application program interface (API) which is known as northbound interface. Control plane communicates with data plane using OpenFlow protocol and it also contains controller. Infrastructure layer consist of different network devices and it will communicate with control layer using OpenFlow protocol, a south bound interface [1].

1.2 OpenFlow Protocol

OpenFlow is an open standard that can be implemented in Ethernet switches, routers and wireless access points (AP) [15]. It is a standard means of communication between control layer and forwarding layer of SDN architecture. OpenFlow is a standard communications interface defined between the control layers and forwarding layer of SDN architecture which we can call it as SDN southbound interface. OpenFlow allows direct access and programmability of forwarding plane devices such as switches, routers and wireless access points. The OpenFlow protocol is implemented on both sides of the interface between network infrastructure devices and the SDN control software. Network flexibility is provided by OpenFlow which describes high level tool support for performing dynamic changes to workflow [16].

OpenFlow-based applications have been proposed to ease the configuration of a network, to simplify network management and to add security features, to virtualize networks and data centers and to deploy mobile systems [17].

OpenFlow switch and controller communicate via OpenFlow protocol. OpenFlow uses the concept of flows to identify network traffic based on pre-defined match-action rules that can be statically or dynamically programmed by the SDN control software. These rules are known as flow rules. OpenFlow protocol defines messages such as packet-received, send-packet-out, modify-routing-table, get-state etc. Data path of OpenFlow switch contains flow table. Each flow table consists of packet fields to match and an action to be taken on packet. If packet is coming for the first time and flow table does not know what to do with this, it will forward that packet to controller. Controller then responsible to deal with that packet and decides on how to handle it. Controller can add an entry to flow table for future incoming packets or it can drop the packet [2].

1.3 SDN Controllers

There are different SDN controllers available to use in the market. They are developed using different languages. i.e. C, Java, Python etc. Here we are considering the Warp controller to create OpenFlow protocol version 1.2. Warp is a SDN controller which uses Apache Avro protocol for serialization library and set of Avro schema files for different versions of OpenFlow protocols. It is developed using Java. Warp is a component based software defined networking framework built on top of Akka. It maintains connection with SDN switches, supports dynamic (re)loading of predefined or user defined components and provides event based way of communication between loaded components and connected switches [3]. All functions of the control and management plane are performed by the controller. It has

full network topology information and the location of hosts and external paths (MAC and routing tables). When a switch receives a packet in which there is no entry in its Flow Table, it forwards the message to the controller asking for the action to take upon this new flow [18]. Given the importance of the OpenFlow controller for the software defined network it directs, it is key to understand the performance and behavior of this important software component for experimenters as well as for operators of productive networks [19].

1.4 Thesis Work

Our aim of this thesis is to contribute to the isolated data structure support of OpenFlow protocol through software infrastructure proposed by Warp development group. Initially Warp controller was supporting only OpenFlow version 1.0 and 1.3, our initiative was to develop OpenFlow version 1.2 schema which will be integrated and supported by Warp controller. As explained in literature survey, it's not easy to make any updates to existing OpenFlow protocol as well as integrate new version of OpenFlow protocol to any controller. Also it is hard to do such changes at run time. Warp development group proposed a solution to it by using Apache Avro which is an open source project supports services like data serialization and data exchange. We have used the same data structure to create an OpenFlow version 1.2 schema by which Warp controller will support all OpenFlow version 1.2 specifications.

1.5 Literature survey

Whenever there is an update in OpenFlow protocol, every controller should support that particular version but it's not that easy as it seems. Controllers are not supporting new versions of OpenFlow protocol in an easy and integrated fashion. So there should be some means by which it can be done in a manner which does not create any problem and supports new version of OpenFlow protocol without any difficulty. As there is a new version of OpenFlow protocol release every year and it already have five active versions i.e. OpenFlow version 1.0, OpenFlow version 1.1, OpenFlow version 1.2, OpenFlow version 1.3 and OpenFlow version 1.4, it increases the complexity of which can cause the roadblocks for the networks, software's and hardware engineers. Each version has complex feature dependencies, unstable version negotiation and lack of state machine definition which is very difficult to deal with. The approach of distilling the core abstraction present in 5 existing versions of OpenFlow and refactored them into a simple API is known as tinyNBI. The goal of tinyNBI is to allow configuration of all existing OpenFlow abstraction without having to deal with unique features of each version of OpenFlow or their level of support of target switches. It defines a low level stack for all 5 OpenFlow versions which provides facilities to manage messages, state machines, system interfaces and configurations. This approach is unsuccessful at keeping the interface simple in the face of multiple versions of OpenFlow and differing capabilities of switches.

There are different programming languages that have been developed to model OpenFlow primitives like Frenetic [6], Maple and tinyNBI is developed using C

language. TinyNBI handles multiple concurrent OpenFlow versions and varying capabilities without placing additional effort on the application developer.

Writing OpenFlow application that operate correctly in robust network environment is a tough proposition. Building high level network abstraction is difficult so tinyNBI provided stable foundation for designing these high level abstraction on top of tinyNBI that allow for clean separation of operator versus developer concern [4].

There is a work related to non-uniformity of capable support across target switches. NOXIS presents a high level interface and defined mechanism for meeting low level details of a target switch with driver development [5].

By the discussion, we see how variable OpenFlow standard's data structure has been and needs to be standardize in easy manner. In our experiment we have used Apache Avro for the data serialization purpose. It provides rich data structures, compact data format, container file to store persistent data and simple integration with dynamic languages. Furthermore, we have used Warp OpenFlow architecture and developed OpenFlow protocol version 1.2 by creating schema with the help of Avro [7].

1.6 Thesis Overview

This thesis has a total of 5 chapters. Chapter 1 includes introduction of SDN, OpenFlow protocol, SDN Controllers, thesis work and literature survey. Chapter 2 explains about the Avro data structure which is used to develop OpenFlow protocol version 1.2 in Warp architecture. It also explains about the Warp architecture and contribution in inclusion of OpenFlow protocol version 1.2. And, the actors in Warp

architecture. Chapter 3, describes OpenFlow protocol, OpenFlow Version 1.0, OpenFlow Version 1.1, OpenFlow Version 1.2, OpenFlow Version 1.3 and OpenFlow Version 1.4 specifications. Chapter 4 contains Warp comparison with other OpenFlow architectures like Floodlight, Ryu etc. Finally, Chapter 5 concludes the thesis work, results and explains about what else could be added to this solution as extensions and future work. At last, in Appendix, the code for OpenFlow version 1.2 is mentioned.

Chapter 2 – Data Structure (Avro) and Warp Architecture

To implement the OpenFlow version 1.2 support for the Warp OpenFlow controller, we have used the existing Warp architecture. We have developed an XML schema for OpenFlow version 1.2 which can be integrated into Warp architecture very easily. Also, this architecture allows us to integrate any update or new version of OpenFlow version very easily.

The experiment consist of tool which allow us to integrate OpenFlow version 1.2 into Warp controller. It consists of Apache Avro which we have used to create a schema. Apache Avro is an open source project and data serialization system.

2.1 Avro

Apache Avro [21] is serialization framework developed as a Hadoop subproject. It provides serialization formats for persistent data and communication between Hadoop nodes. To integrate with dynamic programming languages, code generation is not required to serialize and de-serialize structured data, but schema definitions for Apache Avro are required in order to integrate it with other programming languages such as C++, Java, Python etc. Avro uses JSON for schema definitions. When Avro data is read, the schema is read at run time before reading the data. One of features for Avro is that it enables direct serialization from schema definitions without code generation. It just encodes one field after another and in the order they appear in the schema definition [22].

Avro provides data serialization and data exchange services. serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment. Avro supports schema evaluation feature which allows building less decoupled and more robust system [23]. Schema evaluation is another feature which is used for serialization and deserialization and to see missing values, extra values and modified fields within the schema [24].

Avro provides:

1. Rich data structures.
2. A compact, fast, binary data format.
3. A container file, to store persistent data.
4. Remote procedure call (RPC).
5. Simple integration with dynamic languages.

Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages [7].

2.1.1 Schemas

Avro relies on schemas. When Avro data is read, the schema used when writing it is always present. This permits each datum to be written with no per-value overheads, making serialization both fast and small. This also facilitates use with dynamic, scripting languages, since data, together with its schema, is fully self-describing.

When Avro data is stored in a file, its schema is stored with it, so that files may be processed later by any program. If the program reading the data expects a different schema this can be easily resolved, since both schemas are present.

When Avro is used in RPC, the client and server exchange schemas in the connection handshake. (This can be optimized so that, for most calls, no schemas are actually transmitted.) Since both client and server both have the other's full schema, correspondence between same named fields, missing fields, extra fields, etc. can all be easily resolved.

Avro schemas are defined with JSON (JavaScript Object Notation). This facilitates implementation in languages that already have JSON libraries. Figure 2 shows Avro stores data definition i.e. Schema and data together in one file [7].

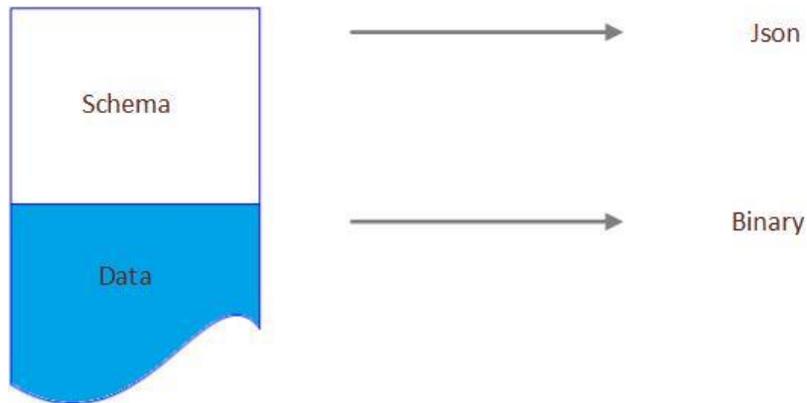


Figure 2: Data Definition (Avro)

2.1.2 Comparison with other systems

Avro provides functionality similar to systems such as Thrift, Protocol Buffers, etc.

Avro differs from these systems in the following fundamental aspects.

Dynamic typing: Avro does not require that code be generated. Data is always accompanied by a schema that permits full processing of that data without code generation, static data types, etc. This facilitates construction of generic data-processing systems and languages.

Untagged data: Since the schema is present when data is read, considerably less type information need be encoded with data, resulting in smaller serialization size.

No manually-assigned field IDs: When a schema changes, both the old and new schema are always present when processing data, so differences may be resolved symbolically, using field names.

One of the feature of Avro is it provides framework for storing and exchanging big data between programs with any languages such as between Java and C, Hive and Python, Pig and Java [7] as shown in figure 3.

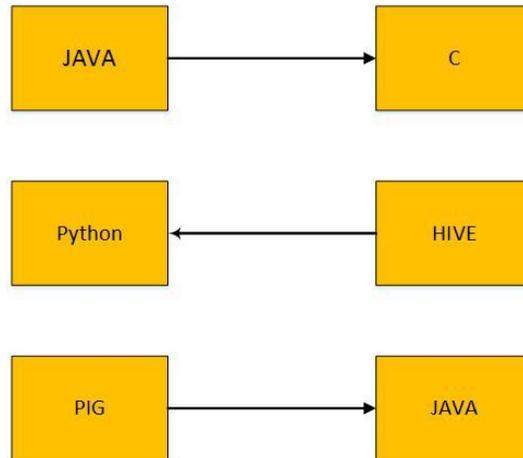


Figure 3: Data exchange between different languages

2.2 Warp Architecture with inclusion of OpenFlow version 1.2

Figure 4 shows all the components of Warp enclosed in red dotted line box. The broken purple line box encloses the Warp OpenFlow driver and the broken green box encloses the Warp OpenFlow controller. The other controllers can use the Warp OpenFlow driver separately. The Rest service provides restful interface to send message to OpenFlow switches that are connected to the controller. SDN applications can interface to the Warp controller using the controller's Java API.

Our contribution to this architecture is an inclusion of OpenFlow version 1.2, .avpr schema file which is placed in Warp OpenFlow driver box. The Warp OpenFlow

version 1.2 is built from modified Avro library and the Avro schema files that describe the OpenFlow protocol. The Avro schema files (.avpr) are kept external to the Warp OpenFlow driver file, so that they can be modified if needed, without needing to rebuild the jar file. When there is a need to modify the Avro file, say to fix a bug or introduce new features, all that needs to be done is to invoke the "init" method of the modified Avro schema file.

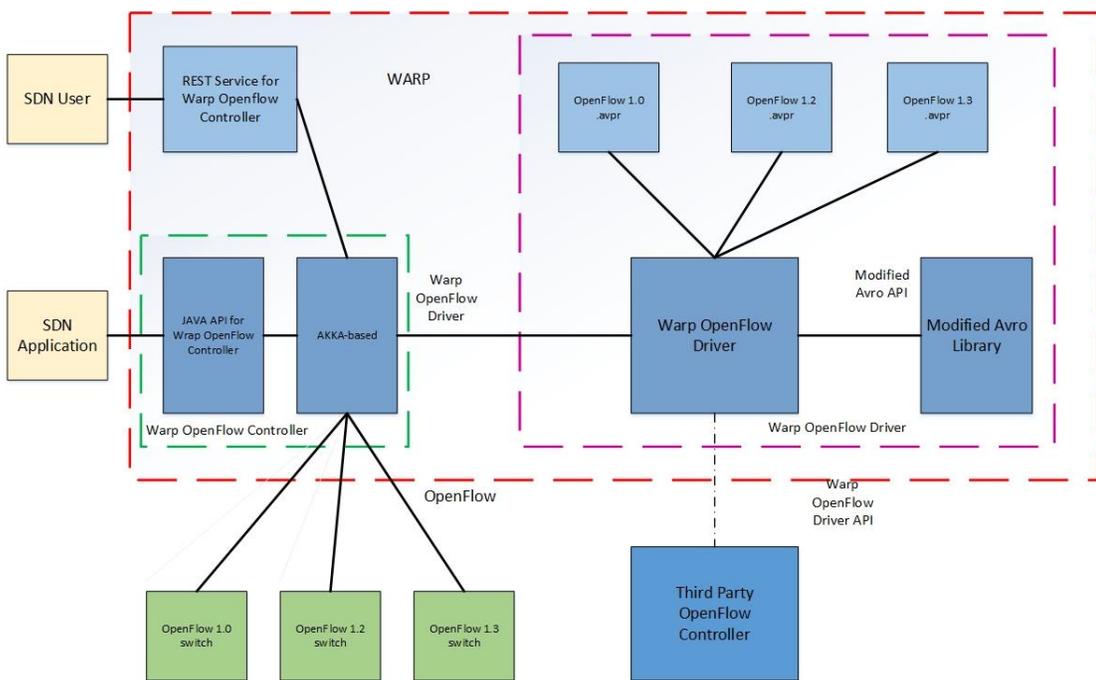


Figure 4: Warp architecture

Warp uses the Akka framework to build a concurrent and scalable controller. Akka is an asynchronous, non-blocking and highly performance event-driven programming model which provides simple and high level abstractions for concurrency and parallelism using actors which are lightweight event driven process [32].

Figure 5 shows the different actors of the Warp controller. The controller itself is the main supervisor of all other actors. TCP listener actor listens the port 6653 which is configurable. When it gets an incoming connection from an OpenFlow switch the TCP listener actor establishes a TCP session launches a OF session handler and OF protocol session handler. The OF protocol session handler (one per switch) handles OpenFlow protocol session related behavior such as exchange of "hello" & "echo" message with switch. The OF session handler is an application developer defined actor. The developer has to encapsulate all reactions to OF events in this actor. Sending OpenFlow messages such as "Flow-mod" messages to the switch is also implemented using events [7].

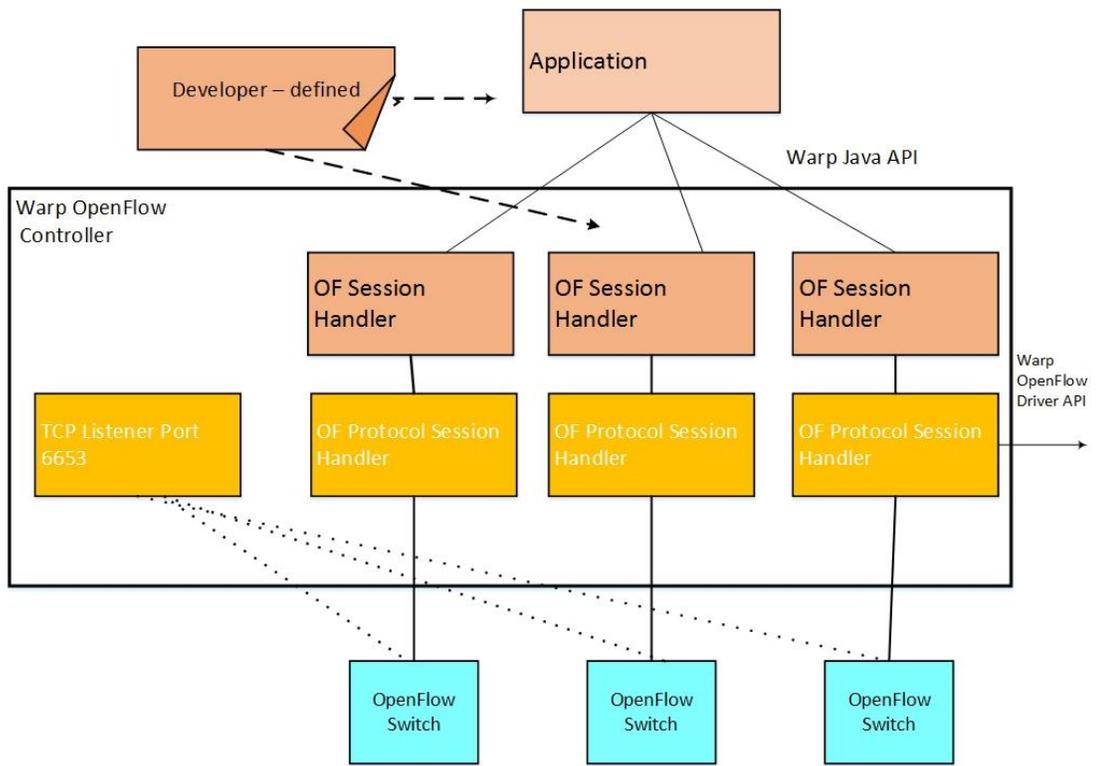


Figure 5: Actors of Warp controller

Chapter 3 – OpenFlow

OpenFlow, an instance of the SDN architecture, is a set of specifications maintained by the Open Networking Forum (ONF). Figure 6 shows overview of OpenFlow protocol. At the core of the specifications is a definition of an abstract packet processing machine, called a switch. The switch processes packets using a combination of packet contents and switch configuration state. A protocol is defined for manipulating the switch's configuration state as well as receiving certain switch events. Finally, a controller is an element that speaks the protocol to manage the configuration state of many switches and respond to events [29].

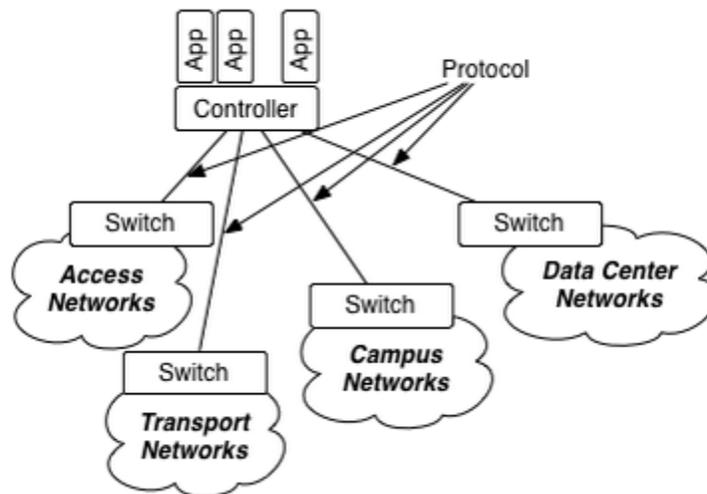


Figure 6: OpenFlow overview

The OpenFlow protocol can be broken into four components: message layer, state machine, system interface, and configuration. The image and table below illustrate these components, their interaction, and describe their function at a high level.

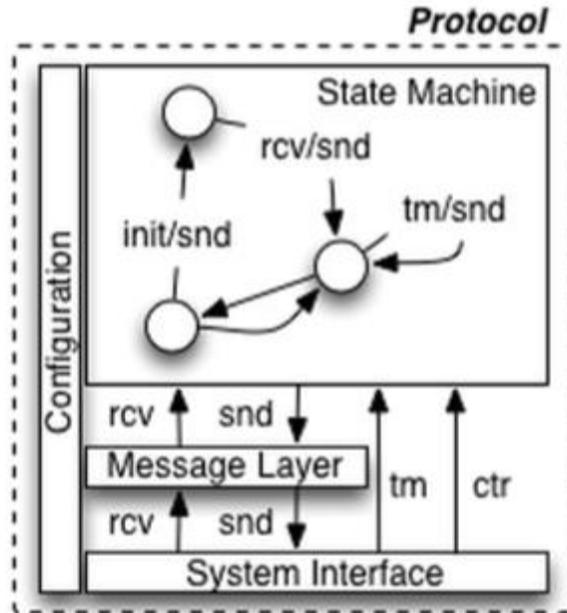


Figure 7: Protocol decomposition

Figure 7 shows different components of protocol decomposition explained as below:

1. Message Layer: The message layer is the core of the protocol stack. It defines valid structure and semantics for all messages. A typical message layer supports the ability to construct, copy, compare, print, and manipulate messages.
2. State Machine: The state machine defines the core low-level behavior of the protocol. Typically, it is used to describe actions such as negotiation, capability discover, flow control, delivery, etc.

3. **System Interface:** The system interface defines how the protocol interacts with the outside world. It typically identifies necessary and optional interfaces along with their intended use, such as TLS and TCP as transport channels.
4. **Configuration:** Almost all aspects of the protocol have configurations or initial values. Configuration can cover anything from default buffer sizes and reply intervals to X.509 certificates.
5. **Data Model:** Another way to consider the OpenFlow protocol is to understand its underlying data model. Each switch maintains a relational data model that contains the attributes for each OpenFlow abstraction. These attributes either describe an abstractions capability, its configuration state, or some set of current statistics.

Every OpenFlow message [figure 8] begins with the same header structure. This fixed structure serves three roles that are independent of the version of OpenFlow being used. First, the version field indicates the version of OpenFlow which this message belongs. Second, the length field indicates where this message will end in the byte stream starting from the first byte of the header. Third, the xid, or transaction identifier, is a unique value used to match requests to responses. The type field which indicates what type of message is present and how to interpret the payload, is version dependent [28].

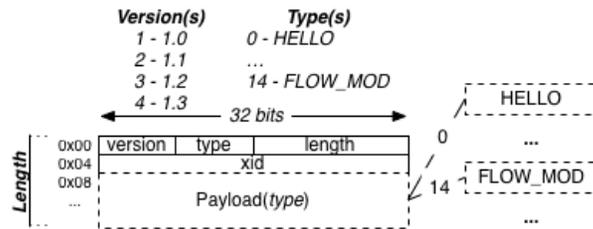


Figure 8: Message Layer

There are different OpenFlow versions i.e. OpenFlow Version 1.0, OpenFlow Version 1.1, OpenFlow Version 1.2 and OpenFlow Version 1.3 and OpenFlow Version 1.4 [20].

3.1 OpenFlow Version 1.0

Following are the specifications of OpenFlow protocol version 1.0 [26].

3.1.1 Slicing

OpenFlow now supports multiple queues per output port. Queues support the ability to provide minimum bandwidth guarantees; the bandwidth allocated to each queue is configurable. The name slicing is derived from the ability to provide a slice of the available network bandwidth to each queue.

3.1.2 Flow Cookies

Flows have been extended to include an opaque identifier, referred to as a cookie. The cookie is specified by the controller when the flow is installed; the cookie will be returned as part of each flow stats and flow expired message.

3.1.3 User-specifiable data path description

The OFPST DESC (switch description) reply now includes a data path description field. This is a user specifiable field that allows a switch to return a string specified by the switch owner to describe the switch.

3.1.4 Match on IP fields in ARP packets

The reference implementation can now match on IP fields inside ARP packets. The source and destination protocol address are mapped to the nw src and nw dst fields respectively, and the opcode is mapped to the nw proto field.

3.1.5 Match on IP ToS/DSCP bits

OpenFlow now supports matching on the IP ToS/DSCP bits.

3.1.6 Querying port stats for individual ports

Port stat request messages include a port_no field to allow stats for individual ports to be queried. Port stats for all ports can still be requested by specifying OFPP_NONE as the port number.

3.1.7 Improved flow duration resolution in stats/expiry messages

Flow durations in stats and expiry messages are now expressed with nanosecond resolution. Note that the accuracy of flow durations in the reference implementation is on the order of milliseconds. (The actual accuracy is in part dependent upon kernel parameters.)

3.1.8 Other changes to the Specification

- remove multi_phy_tx spec text and capability bit
- clarify execution order of actions
- replace SSL refs with TLS
- resolve overlap ambiguity
- clarify flow mod to non-existing port
- clarify port definition
- update packet flow diagram
- update header parsing diagram for ICMP
- fix English ambiguity for flow-removed messages
- fix a sync message English ambiguity
- note that multiple controller support is undefined
- clarify that byte equal's octet
- note counter wrap-around
- removed warning not to build a switch from this specification

3.2 OpenFlow Version 1.1

Following are the specifications of OpenFlow protocol version 1.1 [26].

3.2.1 Multiple Tables

Prior versions of the OpenFlow specification did expose to the controller the abstraction of a single table.

The OpenFlow pipeline could internally be mapped to multiple tables, such as having a separate wildcard and exact match table, but those tables would always act logically as a single table.

OpenFlow 1.1 introduces a more flexible pipeline with multiple tables. Exposing multiple tables has many advantages. The first advantage is that many hardware have multiple tables internally (for example L2 table, L3 table, multiple TCAM lookups), and the multiple table support of OpenFlow may enable to expose this hardware with greater efficiency and flexibility. The second advantage is that many network deployments combine orthogonal processing of packets (for example ACL, QoS and routing), forcing all those processing in a single table creates huge ruleset due to the cross product of individual rules, multiple table may decouple properly those processing.

The new OpenFlow pipeline with multiple table is quite different from the simple pipeline of prior OpenFlow versions. The new OpenFlow pipeline expose a set of completely generic tables, supporting the full match and full set of actions. It's difficult to build a pipeline abstraction that represent accurately all possible hardware, therefore OpenFlow 1.1 is based on a generic and flexible pipeline that may be mapped to the hardware. Some limited table capabilities are available to denote what each table is capable of supporting.

Packet are processed through the pipeline, they are matched and processed in the first table, and may be matched and processed in other tables. As it goes through the pipeline, a packet is associated with an action set, accumulating action, and a generic metadata register. The action set is resolved at the end of the pipeline and applied to the packet. The metadata can be matched and written at each table and enables to carry state between tables.

OpenFlow introduces a new protocol object called instruction to control pipeline processing. Actions which were directly attached to flows in previous versions are now encapsulated in instructions, instructions may apply those actions between tables or accumulate them in the packet action set. Instructions can also change the metadata, or direct packet to another table.

- The switch now expose a pipeline with multiple tables.
- Flow entry have instruction to control pipeline processing
- Controller can choose packet traversal of tables via go to instruction
- Metadata field (64 bits) can be set and match in tables
- Packet actions can be merged in packet action set
- Packet action set is executed at the end of pipeline
- Packet actions can be applied between table stages
- Table miss can send to controller, continue to next table or drop
- Rudimentary table capability and configuration

3.2.2 Groups

The new group abstraction enables OpenFlow to represent a set of ports as a single entity for forwarding packets. Different types of groups are provided, to represent different abstractions such as multicasting or multipathing. Each group is composed of a set group buckets, each group bucket contains the set of actions to be applied before forwarding to the port. Group's buckets can also forward to other groups, enabling to chain groups together.

- Group indirection to represent a set of ports
- Group table with 4 types of groups:
 - All - used for multicast and flooding
 - Select - used for multipath
 - Indirect - simple indirection
 - Fast Failover - use first live port
- Group action to direct a flow to a group
- Group buckets contains actions related to the individual port

3.2.3 MPLS & VLAN

Prior versions of the OpenFlow specification had limited VLAN support, it only supported a single level of VLAN tagging with ambiguous semantic. The new tagging support has explicit actions to add, modify and remove VLAN tags, and can support multiple level of VLAN tagging. It also adds similar support the MPLS shim headers.

- Support for VLAN and QinQ, adding, modifying and removing VLAN headers
- Support for MPLS, adding, modifying and removing MPLS shim headers

3.2.4 Virtual Ports

Prior versions of the OpenFlow specification assumed that all the ports of the OpenFlow switch were physical ports. This version of the specification add support for virtual ports, which can represent complex forwarding abstractions such as LAGs or tunnels.

- Make port number 32 bits, enable larger number of ports
- Enable switch to provide virtual port as OpenFlow ports
- Augment packet-in to report both virtual and physical ports

3.2.5 Controller connection failure

Prior versions of the OpenFlow specification introduced the emergency flow cache as a way to deal with the loss of connectivity with the controller. The emergency flow cache feature was removed in this version of the specification, due to the lack of adoption, the complexity to implement it and other issues with the feature semantic.

This version of the specification add two simpler modes to deal with the loss of connectivity with the controller. In fail secure mode, the switch continues operating in

OpenFlow mode, until it reconnects to a controller. In fail standalone mode, the switch revert to using normal processing (Ethernet switching).

- Remove Emergency Flow Cache from spec
- Connection interruption trigger fail secure or fail standalone mode.

3.2.6 Other changes

- Remove 802.1d-specific text from the specification
- Cookie Enhancements Proposal - cookie mask for filtering
- Set queue action (unbundled from output port action)
- Maskable DL and NW address match fields
- Add TTL decrement, set and copy actions for IPv4 and MPLS
- SCTP header matching and rewriting support
- Set ECN action
- Define message handling: no loss, may reorder if no barrier
- Rename VENDOR APIs to EXPERIMENTER APIs
- Many other bug fixes, rewording and clarifications.

3.3 OpenFlow Version 1.2

Following are the specifications which supports OpenFlow version 1.2 [8]. We have created a data structure (schema) for OpenFlow version 1.2 and integrated it with Warp architecture.

3.3.1 Extensible Match Support

Prior versions of the OpenFlow specification used a static fixed length structure to specify `ofp_match`, which prevents flexible expression of matches and prevents inclusion of new match fields. The `ofp_match` has been changed to a TLV structure, called OpenFlow Extensible Match (OXM), which dramatically increases flexibility. The match fields themselves have been reorganized. In the previous static structure, many fields were overloaded; for example `tcp.src_port`, `udp.src_port`, and `icmp.code` were using the same field entry. Now, every logical field has its own unique type.

3.3.2 Extensible ‘set field’ packet rewriting support

Prior versions of the OpenFlow specification were using hand-crafted actions to rewrite header fields. The Extensible `set_field` action reuses the OXM encoding defined for matches, and enables to rewrite any header field in a single action. This allows any new match field, including experimenter fields, to be available for rewrite. This makes the specification cleaner and eases cost of introducing new fields.

- Deprecate most header rewrite actions
- Introduce generic set-field action
- Reuse match TLV structure (OXM) in set-field action

3.3.3 Extensible context expression in ‘packet-in’

The packet-in message did include some of the packet context (ingress port), but not all (metadata), preventing the controller from figuring how match did happen in the

table and which flow entries would match or not match. Rather than introduce a hard coded field in the packet-in message, the flexible OXM encoding is used to carry packet context.

- Reuse match TLV structure (OXM) to describe metadata in packet-in
- Include the 'metadata' field in packet-in
- Move ingress port and physical port from static field to OXM encoding
- Allow to optionally include packet header fields in TLV structure

3.3.4 Extensible Error messages via experimenter error type

An experimenter error code has been added, enabling experimenter functionality to generate custom error messages. The format is identical to other experimenter APIs.

3.3.5 IPv6 support added

Basic support for IPv6 match and header rewrite has been added, via the Flexible match support.

- Added support for matching on IPv6 source address, destination address, protocol number, traffic class, ICMPv6 type, ICMPv6 code and IPv6 neighbor discovery header fields
- Added support for matching on IPv6 flow label

3.3.6 Simplified behavior of flow-mod request

The behavior of flow-mod request has been simplified.

- MODIFY and MODIFY STRICT commands never insert new flows in the table
- New flag OFPFF RESET COUNTS to control counter reset
- Remove quirky behavior for cookie field.

3.3.7 Removed packet parsing specification

The OpenFlow specification no longer attempts to define how to parse packets. The match fields are only defined logically.

- OpenFlow does not mandate how to parse packets
- Parsing consistency achieved via OXM pre-requisite

3.4 OpenFlow Version 1.3

Following are the specifications of OpenFlow version 1.3 protocol [27].

3.4.1 Refactor capabilities negotiation

Prior versions of the OpenFlow specification included limited expression of the capabilities of an OpenFlow switch. OpenFlow 1.3 include a more flexible framework to express capabilities.

The main change is the improved description of table capabilities. Those capabilities have been moved out of the table statistics structure in its own request/reply message, and encoded using a flexible TLV format. This enables the additions of next-table capabilities, table-miss flow entry capabilities and experimenter capabilities. Other changes include renaming the 'stats' framework into the 'multipart' framework to

reflect the fact that it is now used for both statistics and capabilities, and the move of port descriptions into its own multipart message to enable support of a greater number of ports.

List of features for Refactor capabilities negotiation:

- Rename 'stats' framework into the 'multipart' framework.
- Enable 'multipart' requests (requests spanning multiple messages).
- Move port list description to its own multipart request/reply.
- Move table capabilities to its own multipart request/reply.
- Create flexible property structure to express table capabilities.
- Enable to express experimenter capabilities.
- Add capabilities for table-miss flow entries.
- Add next-table (i.e. goto) capabilities

3.4.2 More flexible table miss support

Prior versions of the OpenFlow specification included table configuration flags to select one of three behavior for handling table-misses (packet not matching any flows in the table). OpenFlow 1.3 replace those limited flags with the table-miss flow entry, a special flow entry describing the behavior on table miss.

The table-miss flow entry uses standard OpenFlow instructions and actions to process table-miss packets, this enables to use the full flexibility of OpenFlow in processing

those packets. All previous behavior expressed by the table-miss config flags can be expressed using the table-miss flow entry. Many new way of handling table-miss, such as processing table-miss with normal, can now trivially be described by the OpenFlow protocol.

- Remove table-miss config flags.
- Define table-miss flow entry as the all wildcard, lowest priority flow entry.
- Mandate support of the table-miss flow entry in every table to process table-miss packets.
- Add capabilities to describe the table-miss flow entry.
- Change table-miss default to drop packets.

3.4.3 IPv6 Extension Header handling support

Add the ability of match the presence of common IPv6 extension headers, and some anomalous conditions in IPv6 extension headers. A new OXM pseudo header field OXM_OF_IPV6_EXTHDR enables to match the following conditions:

- Hop-by-hop IPv6 extension header is present.
- Router IPv6 extension header is present.
- Fragmentation IPv6 extension header is present.
- Destination options IPv6 extension headers is present.
- Authentication IPv6 extension header is present.
- Encrypted Security Payload IPv6 extension header is present.

- No Next Header IPv6 extension header is present.
- IPv6 extension headers out of preferred order.
- Unexpected IPv6 extension header encountered.

3.4.4 Per Flow meters

Add support for per-flow meters. Per-flow meters can be attached to flow entries and can measure and control the rate of packets. One of the main applications of per-flow meters is to rate limit packets sent to the controller. The per-flow meter feature is based on a new flexible meter framework, which includes the ability to describe complex meters through the use of multiple metering bands, metering statistics and capabilities. Currently, only simple rate-limiter meters are defined over this framework. Support for color-aware meters, which support Diff-Serv style operation and are tightly integrated in the pipeline, was postponed to a later release.

- Flexible meter framework based on per-flow meters and meter bands.
- Meter statistics, including per band statistics.
- Enable to attach meters flexibly to flow entries.
- Simple rate-limiter support (drop packets).

3.4.5 Per connection event filtering

Previous version of the specification introduced the ability for a switch to connect to multiple controllers for fault tolerance and load balancing. Per connection event filtering improves the multi-controller support by enabling each controller to filter

events from the switch it does not want. A new set of OpenFlow messages enables a controller to configure an event filter on its own connection to the switch. Asynchronous messages can be filtered by type and reason. This event filter comes in addition to other existing mechanisms that enable or disable asynchronous messages, for example the generation of flow-removed events can be configured per flow. Each controller can have a separate filter for the slave role and the master/equal role.

- Add asynchronous message filter for each controller connection.
- Controller message to set/get the asynchronous message filter.
- Set default filter value to match OpenFlow 1.2 behavior.
- Remove OFPC_INVALID_TTL_TO_CONTROLLER config flag.

3.4.6 Auxiliary connections

In previous version of the specification, the channel between the switch and the controller is exclusively made of a single TCP connection, which does not allow to exploit the parallelism available in most switch implementations. OpenFlow 1.3 enables a switch to create auxiliary connections to supplement the main connection between the switch and the controller. Auxiliary connections are mostly useful to carry packet-in and packet-out messages.

- Enable switch to create auxiliary connections to the controller.
- Mandate that auxiliary connection cannot exist when main connection is not alive.

- Add auxiliary-id to the protocol to disambiguate the type of connection.
- Enable auxiliary connection over UDP and DTLS.

3.4.7 MPLS BoS matching

A new OXM field OXM_OF_MPLS_BOS has been added to match the Bottom of Stack bit (BoS) from the MPLS header. The BoS bit indicates if other MPLS shim header are in the payload of the present MPLS packet, and matching this bit can help to disambiguate case where the MPLS label is reused across levels of MPLS encapsulation.

3.4.8 Provider Backbone Bridging tagging

Add support for tagging packet using Provider Backbone Bridging (PBB) encapsulation. This support enables OpenFlow to support various network deployment based on PBB, such as regular PBB and PBB-TE.

- Push and Pop operation to add PBB header as a tag.
- New OXM field to match I-SID for the PBB header.

3.4.9 Rework tag order

In previous version of the specification, the final order of tags in a packet was statically specified. For example, a MPLS shim header was always inserted after all VLAN tags in the packet. OpenFlow 1.3 removes this restriction, the final order of

tags in a packet is dictated by the order of the tagging operations, and each tagging operation adds its tag in the outermost position.

- Remove defined order of tags in packet from the specification.
- Tags are now always added in the outermost possible position.
- Action-list can add tags in arbitrary order.
- Tag order is predefined for tagging in the action-set.

3.4.10 Tunnel-ID metadata

The logical port abstraction enables OpenFlow to support a wide variety of encapsulations. The tunnel-id metadata `OXM_OF_TUNNEL_ID` is a new OXM field that expose to the OpenFlow pipeline metadata associated with the logical port, most commonly the de-multiplexing field from the encapsulation header.

For example, if the logical port perform GRE encapsulation, the tunnel-id field would map to the GRE key field from the GRE header. After de-capsulation, OpenFlow would be able to match the GRE key in the tunnel-id match field. Similarly, by setting the tunnel-id, OpenFlow would be able to set the GRE key in an encapsulated packet.

3.4.11 Cookies in packet-in

A cookie field was added to the packet-in message. This field takes its value from the flow and sends the packet to the controller. If the packet was not sent by a flow, this

field is set to 0xffffffff. Having the cookie in the packet-in enables the controller to more efficiently classify packet-in, rather than having to match the packet against the full flow table.

3.4.12 Duration for stats

A duration field was added to most statistics, including port statistics, group statistics, queue statistics and meter statistics. The duration field enables to more accurately calculate packet and byte rate from the counters included in those statistics.

3.4.13 On-demand flow counters

New flow-mod flags have been added to disable packet and byte counters on a per-flow basis. Disabling such counters may improve flow handling performance in the switch.

3.4.14 Other changes

- Fix a bug describing VLAN matching.
- Flow entry description now mention priority.
- Flow entry description now mention timeout and cookies.
- Unavailable counters must now be set to all 1.
- Correctly refer to flow entry instead of rule.
- Many other bug fixes, rewording and clarifications.

3.5 OpenFlow Version 1.4

Following are the specifications of OpenFlow protocol version 1.4 [26].

3.5.1 More extensible wire protocol

The OpenFlow protocol was initially designed with many static fixed structures and limited extensibility. The introduction of the OpenFlow Extensible Match (OXM) in version 1.2 added much needed extensibility in the OpenFlow classifier. In version 1.4, many other parts of the protocol have been retrofitted with TLV structures for improved extensibility. This TLV work affected many areas of the protocol. New TLVs have been added in previously fixed structures in the form of properties at the end of the structure. In some areas, the existing TLVs have been changed to use the common property TLV format. TLVs rules have been clarified. This additional extensibility of the protocol will allow a much easier way to add new features to the protocol in the future, and also greatly extend the Experimenter Extension API.

- Port structures: add port description properties, add port mod properties and add port stats properties.
- Table structures: add table mod properties, add table descriptions multipart, add table status asynchronous message.
- Queue structures: migrate queue description to multipart, convert queue description properties to standardized TLVs and add queue stats properties.
- Set-async structures: convert set-async-config to TLVs, add set-async experimenter property.

- Instruction structures: clarify instruction TLVs.
- Actions structures: clarify actions TLVs.
- Experimented structures: clarify experimenter TLVs.
- Properties errors: add a set of unified error codes for all properties.

3.5.2 More descriptive reasons for packet-in

The OpenFlow pipeline saw extensive changes since 1.0, however, the reason values in the *ofp_packet_in* messages did not change. As a result, many distinct parts of the pipeline are using the same reason value. Version 1.4 introduces more descriptive reasons, so that the controller can properly distinguish which part of the pipeline redirected the packet to the controller. The main change is that the "output action" reason OFPR_ACTION is effectively split into four reasons, "apply-action", "action-set", "group bucket" and "packet-out", representing the four distinct context where this action is used. The "no match" reason OFPR_NO_MATCH is renamed to properly reflect the fact that it is generated by the table miss flow entry. The new set of reason values for *ofp_packet_in* message is:

- OFPR_TABLE_MISS: No matching flow (table-miss flow entry).
- OFPR_APPLY_ACTION: Output to controller in apply-actions.
- OFPR_INVALID_TTL: Packet has invalid TTL.
- OFPR_ACTION_SET: Output to controller in action set.
- OFPR_GROUP: Output to controller in group bucket.

- OFPR_PACKET_OUT: Output to controller in packet-out.

3.5.3 Optical port properties

A new set of port properties add support for Optical ports, they include fields to configure and monitor the transmit and receive frequency of a laser, as well as its power. Those new properties can be used to configure and monitor either Ethernet optical port or optical ports on circuit switches.

- Optical port mod property *ofp_port_mod_prop_optical* to configure optical ports.
- Optical port stats property *ofp_port_stats_prop_optical* to monitor optical ports.
- Optical port description property *ofp_port_desc_prop_optical* to describe optical port capabilities.

3.5.4 Flow- removed reason for meter delete

Add a new reason value OFPRR_METER_DELETE for the *ofp_flow_removed* message to denote that the flow entry was removed as a result of a meter deletion.

When a meter is deleted on the switch, all the flow entries that use that meter are removed. This is similar to how group operates. The flow that are removed may

generate a flow-removed message (depending on config). In version 1.3, we did not have a reason for this condition, version 1.4 fixes that bug.

3.5.5 Flow monitoring

The OpenFlow protocol defines a multi-controller scheme where multiple controller can manage a switch. Flow monitoring allows a controller to monitor in real time the changes to any subsets of the flow table done by other controllers.

The flow monitoring framework allows a controller to define a number of monitors, each selecting a subset of the flow tables. Each monitor includes a table id and a match pattern that defines the subset monitored. When any flow entry is added, modified or removed in one of the subset defined by a flow monitor, an event is sent to the controller to inform it of the change.

- Multipart request *ofp_flow_monitor_request* to set flow monitors on the switch.
- Flow monitor update events sent to controller, with full details using *ofp_flow_update_full* or abbreviated using *ofp_flow_update_abbrev*.
- Monitor flags to define the format of the updates.
- Flow control mechanism to avoid backlog of monitor updates.

3.5.6 Role status event

Version 1.2 of the specification added the ability for a controller to set its role in a multi-controller environment. When a controller elected itself to “master” role, the previous master controller is demoted to “slave” role, however that controller was not informed about it. In version 1.4, the Role Status message enable the switch to inform a controller about change to its role.

- Role status event *OFPT_ROLE_STATUS* to inform controller to change to role.
- Role status properties for experimenter data, *ofp_role_prop_experimenter*.

3.5.7 Eviction

Most flow tables have finite capacity. In previous versions of the specification, when a flow table is full, new flow entries are not inserted in the flow table and an error is returned to the controller. However, reaching that point is pretty problematic, as the controller need time to operate on the flow table and this may cause a disruption of service. Eviction adds a mechanism enabling the switch to automatically eliminate entries of lower importance to make space for newer entries. This enables to smoother degradation of behavior when the table is full.

- Table-mod flag *OFPTC_EVICTON* to enable or disable eviction on a table.
- Flow-mod importance to optionally denote the importance of a flow entry for eviction.

- Table-desc eviction property *ofp_table_mod_prop_eviction* to describe the type of eviction performed by the switch.

3.5.8 Vacancy events

Most flow tables have finite capacity. In previous versions of the specification, when a flow table is full, new flow entries are not inserted in the flow table and an error is returned to the controller. However, reaching that point is pretty problematic, as the controller need time to operate on the flow table and this may cause a disruption of service. Vacancy events adds a mechanism enabling the controller to get an early warning based on a capacity threshold chosen by the controller. This allows the controller to react in advance and avoid getting the table full.

- Table status event *OFPT_TABLE_STATUS* with reasons *OFPTR_VACANCY_DOWN* and *OFPTR_VACANCY_UP* to inform controller of vacancy change.
- Hysteresis mechanism to avoid spurious events using two threshold, *vacancy_down* and *vacancy_up*.
- Table-mod vacancy property to set vacancy thresholds, *ofp_table_mod_prop_vacancy*.

3.5.9 Bundles

Add the bundle mechanism, enabling to apply a group of OpenFlow message as a single operation. This enables the quasi-atomic application of related changes, and to better synchronize changes across a series of switches.

- Bundle control message *OFPT_BUNDLE_CONTROL* to create, destroy and commit bundles.
- Bundle add message *OFPT_BUNDLE_ADD_MESSAGE* to add an OpenFlow message into a bundle.
- Bundle error type *OFPET_BUNDLE_FAILED* to report bundle operation errors.

3.5.10 Synchronized tables

Many switches can perform multiple lookups on the same lookup data. For example, a standard Ethernet learning table performs a learning lookup and a forwarding lookup on the same set of MAC addresses. Another example is RPF checks which reuses the IP forwarding data. The synchronized table feature enables to represent those constructs as a set of two tables which data is synchronized. Synchronized table is expressed using a new property in the table feature, *OFPTFPT_TABLE_SYNC_FROM*. It defines the synchronization abstraction between the two flow tables, however it does not define and express the flow entry transformation between the flow tables.

3.5.11 Group and meter change notifications

The OpenFlow protocol defines a multi-controller scheme where multiple controller can manage a switch. Group and Meter change notifications allow a controller to monitor in real time the changes to the group table or meter table done by other controllers.

The “group-mod” and “meter-mod” requests are simple encapsulated in an *OFPT_REQUESTFORWARD* asynchronous message sent to other controllers. Those notifications are enabled and disabled via the “setasync-config” message.

3.5.12 Error code and bad priority

Some switches may have restrictions on the priorities that can be used in a table. For example a switch may be enforcing some “longest prefix match” rule in a table, requiring the priority to be related to the mask. A new error code, *OFPFMFC_BAD_PRIORITY*, enables the switch to properly inform the controller when this happens.

3.5.13 Error code and set-async-config

The *OFPT_GET_ASYNC_REQUEST* feature was introduced in version 1.3.0. There was no error messages defined for that features, however it is possible for this request

to fail. A new error type, *OFPET_ASYNC_CONFIG_FAILED*, which appropriate code, enables the switch to properly inform the controller when this happens.

The set of error codes defined for *OFPET_ASYNC_CONFIG_FAILED* are:

- *OFACFC_INVALID*: One mask is invalid.
- *OFACFC_UNSUPPORTED*: Requested configuration not supported.
- *OFACFC_EPERM*: Permissions error.

3.5.14 PBB UCA header field

A new OXM field *OFXMT_OFB_PBB_UCA* has been added to match the “use customer address” header field from the PBB header.

3.5.15 Error code for duplicate instruction

The OpenFlow specification defines the instructions included in a flow entry as a set, and that an instruction cannot be duplicated in that set. A new error code, *OFBIC_DUP_INST*, enables the switch to properly inform the controller when flow entries contain duplicate instructions.

3.5.16 Error code for multipart timeout

Multipart request and replies are encoded as a sequence of messages. This version of the specification defines how to deal with unterminated sequence, some minimum timeout are defined as well as error codes.

- Define minimum timeout (100 ms) and error code (*OFPBRC_MULTIPART_REQUEST_TIMEOUT*) for unterminated multipart request sequences.
- Define minimum timeout (1 s) and error code (*OFPBRC_MULTIPART_REPLY_TIMEOUT*) for unterminated multipart reply sequences.

3.5.17 Change default TCP port to 6653

IANA allocated to ONF the TCP port number 6653 to be used by the OpenFlow switch protocol. All uses of the previous port numbers, 6633 and 976, should be discontinued. OpenFlow switches and OpenFlow controllers must use 6653 by default (when not using a user specified port number).

Chapter 4 – Warp comparison with other OpenFlow Controllers

Warp architecture relies on Avro schema. OpenFlow protocol is defined in Avro schema file which consists of fields contained in header format of OpenFlow messages.

Warp supports OpenFlow version 1.0, OpenFlow version 1.2 and OpenFlow version 1.3. Each version is represented by separate schema files. Schema file defines static data type and constraints.

This chapters clearly explain about how Avro data structure has upper hand than the other existing data structures comparing factors which could heavily impact the performance of OpenFlow controller architecture.

Following are the factors used for comparison of Warp and Floodlight, Ryu controller architectures.

4.1 Integrate new version of OpenFlow

As Avro stores data definition i.e. schema and data together, while data is read, it will already have schema present which reduces the process of translating data structure or object state into a format that can be stored and reconstructed later in same or other computer environment. Also less type information need to be encoded with data. When we want to integrate new version of OpenFlow in Warp, we can either use schema already present for older version and update it with new features or we can create new schema file which will have all required features' header format field.

In case of other OpenFlow controllers like floodlight, Open daylight, we have to create a different class files for different features of OpenFlow and rebuild the source code file every time we integrate the new version of OpenFlow.

So, the basic difference between Warp and other OpenFlow controller's architecture is that we don't have to rebuild the source code file every time we integrate the new version of OpenFlow which will reduce a lot of overhead.

4.2 Update existing version/Run time changes

When we have to update any version of OpenFlow in Warp, we don't have to compile the code every time we make changes in schema file. Data is always accompanied by a schema that permits full processing of that data without code generation, static data types, etc. This facilitates construction of generic data-processing systems and languages. We just have to invoke "init" method in order to make run time changes in the schema. This will initialize all the new variables in the schema. Other OpenFlow controller architecture does not support this feature as if we want to do any changes in code we have to compile the code every time before making it executable.

4.3 Version control

When an Avro schema changes, the old schema contents are still present in the schema file so differences may be resolved symbolically, using field names. This states that version control is very easy as we can easily differentiate the old data and newly added data. Other OpenFlow controller architecture provides OpenFlow support using code generated in different languages. Every time we update the code

new file is generated in order to maintain the versions. If we update in the same file we cannot differentiate old and new data contents. It is difficult to maintain such version files.

4.4 Data exchange

One of the features of Avro, it provides framework for storing and exchanging big data between programs written in different languages such as Java and C, Hive and Python, Pig and Java. As Warp architecture has schema file for different OpenFlow versions, the logic defined to access the schemas can be written in any languages and schemas can be accessed via Avro API.

4.5 Code processing

Figure 7.1 shows the processing of source code and schema respectively. Source code processing contains different stages before it gets ready to use.

In figure 9 (a), source code is given to the preprocessor which will process it and give source code w/substitutions to parser. Parser will parse the source code and create parse tree and give this input to translator. Translator will translate it to assembly code and give this input to assembler. In assembly stage, assembly code is directly translated into machine instructions. Some minimal pre-processing, padding and instruction reordering can occur in this stage. In linking, for each function our program calls, the assembly to that function is actually included in the executable file. Linkers output is .exe file which is an executable.

In figure 9 (b), when we have schema file and wants to update it or integrate a new file into architecture, we just have to call “init” method of that particular schema definition.

This clearly shows schema processing requires less steps and overhead.

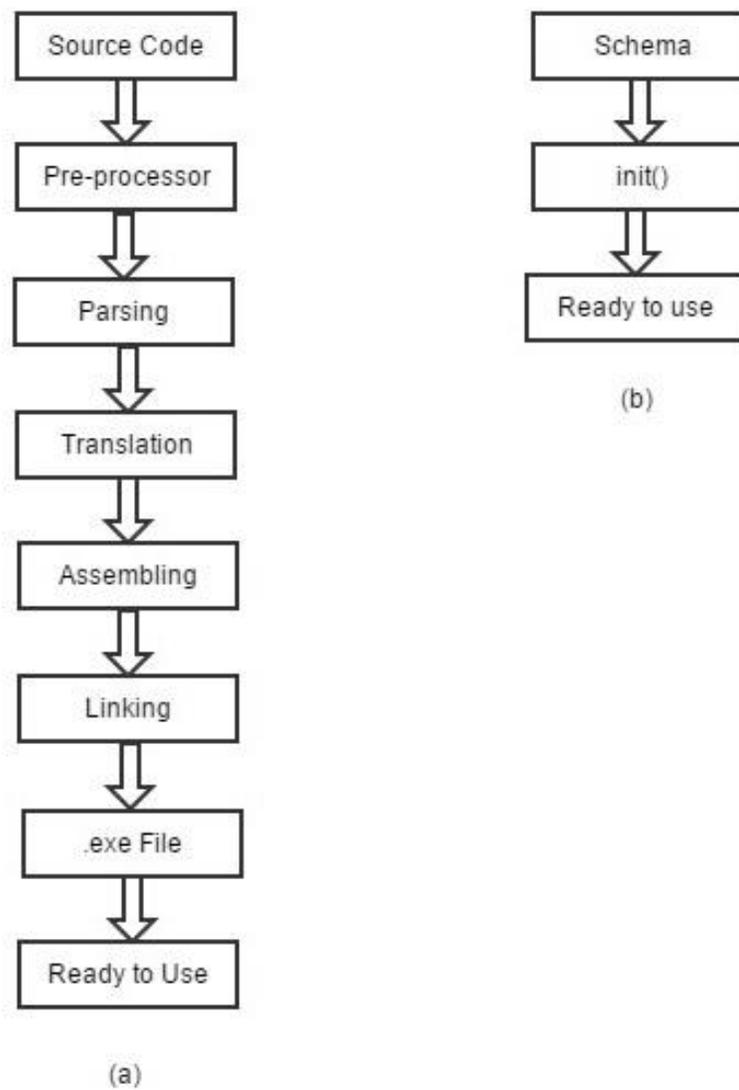


Figure 9: (a) Compilation process of source code (b) Processing of schema

4.6 Architecture Comparison

	Warp	Floodlight	Ryu
File type	schema	java code	python code
Data Serialization	Avro	JSON	Pickle
Data Exchange	Yes	No	No
Version Update	Easy	Difficult	Difficult
Version Control	Easy	Difficult	Difficult
Integration Time	Less	More	More
Run-Time Changes	Possible	Not possible	Not possible
Storage and CPU Utilization for serialization scheme	Less	More	More
Code processing stages	Less	More	More
Compression	Best	Good	Good
Read/Write data	Schema needed	Schema not needed	Schema not needed
Serialization speed	Slow	Medium	Medium

Table 1: Comparison of OpenFlow architectures

The major factors which could impact the performance of the architectures are considered here for comparison. As we can clearly see in the comparison table, Warp has more positive points/factors than other OpenFlow architectures (Floodlight and Ryu)

If we consider 10 major factors of comparison between all, we see 8 factors are in favor of Warp. If we take the ratio of positive factors to total number of factors considered we can clearly mention which is the more flexible architecture.

The result we get after taking the ratios is as follows:

	Warp	Floodlight	Ryu
Flexibility (%)	80	20	20

Table 2: Flexibility (%)

Table 2 clearly shows that the flexibility of Warp is considerably greater as compared to Floodlight and Ryu.

Chapter 5 – Results, Conclusion and Future Work

We have successfully wrote a schema for OpenFlow version 1.2 and integrated it with Warp SDN architecture. As there is isolated data structure support of OpenFlow protocol, we have used software infrastructure proposed by Warp development group which uses Avro which provides rich data structures support. We also compare Warp, Floodlight and Ryu to show how flexible Warp architecture is.

Now, Warp SDN architecture supports OpenFlow versions 1.0, 1.2 and 1.3. We could easily write schema for other OpenFlow versions which are not supported by Warp architecture and also do periodic updates to existing versions of OpenFlow protocols supported by Warp if any with the help of Warp architecture.

References

[1] SDN Definition: <https://www.opennetworking.org/sdn-resources/sdn-definition>

[2] OpenFlow: <http://archive.openflow.org/wp/learnmore/>

[3] Warp: <https://github.com/FlowForwarding/Warp>

[4] tinyNBI: Distilling an API from essential OpenFlow abstractions

[5] NOSIX: A Lightweight Portability Layer for the SDN OS

[6] Frentic: A network Programming language

[7] Avro: <http://avro.apache.org/docs/current/>

[8] OpenFlow Switch Specification version 1.2, Open Networking Foundation

[9] Software Defined Networking: <http://kimia.fi/papers/sdn.pdf>

[10] A survey of Software Defined Networking:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6739370>

[11] Software Defined Networking:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6386859>

[12] Towards secure and dependable Software Defined Networking.

[13] Considerations of Software Defined Networking:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6496914>

[14] Software Defined Networking: A comprehensive Survey

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6994333>

[15] OpenFlow switching: Data plane performance

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5502016>

[16] Flexible workflow management in OpenFlow systems:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=950425>

[17] Network Innovation using OpenFlow:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6587999>

[18] Comparing OpenFlow controller paradigm scalability:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6531863>

[19] A flexible OpenFlow controller benchmark:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6385047>

[20] OpenFlow version specifications

[21] Apache Sqoop Cookbook.

[22] Comparative Survey of Object Serialization Techniques and the Programming Supports:

<http://www.davidpublishing.com/davidpublishing/Upfile/9/6/2012/2012090684992801.pdf>

[23] <https://www.dataxu.com/three-reasons-why-apache-avro-data-serialization-is-a-good-choice-for-openrtb/>

[24] <https://github.com/rfoldes/Avro-Test>

[25] OpenFlow: Enabling Innovation in Campus Networks

[26] “OpenFlow Switch Specification”, version 1.0.0.

<http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf>

[27] “OpenFlow Switch Specification”, version 1.0.0.

<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>

[28] Message Layer

<http://flowgrammable.org/sdn/openflow/message-layer/>

[29] OpenFlow overview

http://flowgrammable.org/sdn/openflow/#tab_protocol

[30] <http://blog.cloudera.com/blog/2011/05/three-reasons-why-apache-avro-data-serialization-is-a-good-choice-for-openrtb/>

[31] <http://www.quora.com/What-are-pros-and-cons-of-Apache-Avro>

[32] Warp openflow controller and driver - Infoblox

Appendix: A – Code

OpenFlow version 1.2

```
{ "namespace" : "of12",
  "protocol" : "ofp12",
  "types" : [
    { "name" : "uint_8", "type" : "fixed", "size" : 1 },
    { "name" : "uint_16", "type" : "fixed", "size" : 2 },
    { "name" : "uint_24", "type" : "fixed", "size" : 3 },
    { "name" : "uint_32", "type" : "fixed", "size" : 4 },
    { "name" : "uint_48", "type" : "fixed", "size" : 6 },
    { "name" : "uint_64", "type" : "fixed", "size" : 8 },
    { "name" : "uint_128", "type" : "fixed", "size" : 16 },
    { "name" : "pad_7", "type" : "fixed", "size" : 7 },
    { "name" : "pad_6", "type" : "fixed", "size" : 6 },
    { "name" : "pad_4", "type" : "fixed", "size" : 4 },
    { "name" : "pad_3", "type" : "fixed", "size" : 3 },
    { "name" : "pad_2", "type" : "fixed", "size" : 2 },
    { "name" : "pad_1", "type" : "fixed", "size" : 1 },

    { "name" : "ofp_type",
      "type" : "enum",
      "items" : "uint_8",
      "list" : [
        { "name" : "OFPT_HELLO", "default" : [0] },
        { "name" : "OFPT_ERROR", "default" : [1] },
        { "name" : "OFPT_ECHO_REQUEST", "default" : [2] },
        { "name" : "OFPT_ECHO_REPLY", "default" : [3] },

        { "name" : "OFPT_FEATURES_REQUEST", "default" : [5] },
        { "name" : "OFPT_FEATURES_REPLY", "default" : [6] },
        { "name" : "OFPT_GET_CONFIG_REQUEST", "default" : [7] },
        { "name" : "OFPT_GET_CONFIG_REPLY", "default" : [8] },
        { "name" : "OFPT_SET_CONFIG", "default" : [9] },

        { "name" : "OFPT_PACKET_IN", "default" : [10] },
        { "name" : "OFPT_FLOW_REMOVED", "default" : [11] },
```

```

{"name":"OFPT_PORT_STATUS", "default":[12]},

{"name":"OFPT_PACKET_OUT", "default":[13]},
{"name":"OFPT_FLOW_MOD", "default":[14]},
{"name":"OFPT_PORT_MOD", "default":[16]},

{"name":"OFPT_BARRIER_REQUEST", "default":[20]},
{"name":"OFPT_BARRIER_REPLY", "default":[21]},

{"name":"OFPT_ROLE_REQUEST", "default":[24]},
{"name":"OFPT_ROLE_REPLY", "default":[25]},

{"name":"OFPT_GET_ASYNC_REQUEST", "default":[26]},
{"name":"OFPT_GET_ASYNC_REPLY", "default":[27]},
{"name":"OFPT_SET_ASYNC", "default":[28]},

{"name":"OFPT_METER_MOD", "default":[29]}
]
},

{"name":"ofp_length",
"type":"enum",
"items":"uint_16",
"list":[
{"name":"OFPL_HELLO_LEN", "default":[0,8]},
{"name":"OFPL_ERROR_LEN", "default":[0,16]},
{"name":"OFPL_ECHO_REQUEST_LEN", "default":[0,8]},
{"name":"OFPL_ECHO_REPLY_LEN", "default":[0,8]},
{"name":"OFPL_EXPERIMENTER_LEN", "default":[0,16]},
{"name":"OFPL_FEATURES_REQUEST_LEN", "default":[0,8]},
{"name":"OFPL_FEATURES_REPLY_LEN", "default":[0,32]},
{"name":"OFPL_GET_CONFIG_REQUEST_LEN", "default":[0,8]},
{"name":"OFPL_GET_CONFIG_REPLY_LEN", "default":[0,12]},
{"name":"OFPL_SET_CONFIG_LEN", "default":[0,12]},
{"name":"OFPL_PACKET_IN_LEN", "default":[0,32]},
{"name":"OFPL_FLOW_REMOVED_LEN", "default":[0,56]},
{"name":"OFPL_PORT_STATUS_LEN", "default":[0,80]},
{"name":"OFPL_PACKET_OUT_LEN", "default":[0,24]},
{"name":"OFPL_FLOW_MOD_LEN", "default":[0,56]},
{"name":"OFPL_GROUP_MOD_LEN", "default":[0,16]},
{"name":"OFPL_PORT_MOD_LEN", "default":[0,40]},
{"name":"OFPL_TABLE_MOD_LEN", "default":[0,16]},

{"name":"OFPL_BARRIER_REQUEST_LEN", "default":[0,24]},

```

```

{"name":"OFPL_BARRIER_REPLY_LEN", "default":[0,24]},

{"name":"OFPL_ROLE_REQUEST_LEN", "default":[0,24]},
{"name":"OFPL_ROLE_REPLY_LEN", "default":[0,24]},
{"name":"OFPL_GET_ASYNC_REQUEST_LEN", "default":[0,8]},
{"name":"OFPL_GET_ASYNC_REPLY_LEN", "default":[0,32]},
{"name":"OFPL_SET_ASYNC_LEN", "default":[0,32]},
{"name":"OFPL_METER_MOD_LEN", "default":[0,16]}
]
},

{"name":"ofp_header",
"type":"record",
"fields":[
{"name":"version", "type":"uint_8", "default":[4]},
{"name":"type", "type":"ofp_type", "default":"OFPT_HELLO"},
{"name":"length", "type":"ofp_length", "default":"OFPL_HELLO_LEN"},
{"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
]
},

{"name":"ofp_hello_header",
"type":"record",
"fields":[
{"name":"version", "type":"uint_8", "default":[4]},
{"name":"type", "type":"ofp_type", "default":"OFPT_HELLO"},
{"name":"length", "type":"ofp_length", "default":"OFPL_HELLO_LEN"},
{"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
]
},

{"name":"ofp_hello_elem_type",
"type":"enum",
"items":"uint_16",
"list": [
{"name":"OFPHET_VERSIONBITMAP", "default":[0,1]}
]
},

{"name":"hello_elem_type_len",
"type":"enum",
"items":"uint_16",
"list": [

```

```

    {"name":"HEL_VERSIONBITMAP", "default":[0,8]}
  ]
},

{"name":"ofp_hello_elem_header",
 "type":"record",
 "fields":[
  {"name":"type", "type":"ofp_hello_elem_type"},
  {"name":"length", "type":"hello_elem_type_len"}
 ]
},

{"name":"ofp_hello",
 "type":"record",
 "fields":[
  {"name":"header", "type":"ofp_hello_header"}
 ]
},

{"name":"ofp_switch_features_request_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_FEATURES_REQUEST"},
  {"name":"length", "type":"ofp_length",
"default":"OFPL_FEATURES_REQUEST_LEN"},
  {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"ofp_switch_features_request",
 "type":"record",
 "fields":[
  {"name":"header", "type":"ofp_switch_features_request_header"}
 ]
},

{"name":"switch_features_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_FEATURES_REPLY"},
  {"name":"length", "type":"ofp_length",
"default":"OFPL_FEATURES_REPLY_LEN"},

```

```

    {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
  ]
},

{"name":"ofp_switch_features",
 "type":"record",
 "fields":[
  {"name":"header", "type":"switch_features_header"},
  {"name":"datapath_id", "type":"uint_64"},
  {"name":"n_buffers", "type":"uint_32"},
  {"name":"n_tables", "type":"uint_8"},
  {"name":"auxiliary_id", "type":"uint_8"},
  {"name":"pad", "type":"pad_2"},
  {"name":"capabilities", "type":"uint_32"},
  {"name":"reserved", "type":"uint_32"}
 ]
},

{"name":"ofp_set_switch_config_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_SET_CONFIG"},
  {"name":"length", "type":"ofp_length", "default":"OFPL_SET_CONFIG_LEN"},
  {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"ofp_set_switch_config",
 "type":"record",
 "fields":[
  {"name":"header", "type":"ofp_set_switch_config_header"},
  {"name":"flags", "type":"uint_16"},
  {"name":"miss_send_len", "type":"uint_16"}
 ]
},

{"name":"get_config_request_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type",
"default":"OFPT_GET_CONFIG_REQUEST"},
 "type":"ofp_type",

```

```

    {"name":"length",
"default":"OFPL_GET_CONFIG_REQUEST_LEN"},
    {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
  ]
},

{"name":"ofp_get_config_request",
"type":"record",
"fields":[
  {"name":"header", "type":"get_config_request_header"}
]
},

{"name":"ofp_config_flags",
"type":"enum",
"items":"uint_16",
"list":[{"name":"OFPC_FRAG_NORMAL", "default":[0, 0]},
  {"name":"OFPC_FRAG_DROP", "default":{"shift":[1, 0]}},
  {"name":"OFPC_FRAG_REASM", "default":{"shift":[1, 1]}},
  {"name":"OFPC_FRAG_MASK", "default":[0, 3]}
]
},

{"name":"switch_config_header",
"type":"record",
"fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_GET_CONFIG_REPLY"},
  {"name":"length",
"default":"OFPL_GET_CONFIG_REPLY_LEN"},
    {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
  ]
},

{"name":"ofp_switch_config",
"type":"record",
"fields":[
  {"name":"header", "type":"switch_config_header"},
  {"name":"flags", "type":"uint_16"},
  {"name":"miss_send_len", "type":"uint_16"}
]
},

{"name":"ofp_match_type",

```

```

    "type": "enum",
    "items": "uint_16",
    "list": [{"name": "OFPMT_STANDARD", "default": [0,0]},
             {"name": "OFPMT_OXM", "default": [0,1]}]
  },

  {"name": "ofp_match_length",
   "type": "enum",
   "items": "uint_16",
   "list": [{"name": "OFPML_EMPTY", "default": [0,4]},
            {"name": "OFPML_IN_PORT", "default": [0,8]},
            {"name": "OFPFF_IN_PHY_PORT", "default": [0,8]}]
  },

  {"name": "oxm_tlv_header_ingress_port",
   "type": "bitmap",
   "size": 32,
   "default": {"or": [{"shift": [32768, 16]}, {"shift": [0, 9]}, {"shift": [0, 8]}, 4]}
  },

  {"name": "oxm_tlv_ingress_port",
   "type": "record",
   "fields": [
     {"name": "header", "type": "oxm_tlv_header_ingress_port"},
     {"name": "tlv", "type": "uint_32", "default": [0,0,0,0]}
   ]
  },

  {"name": "oxm_tlv_header_in_phy_port",
   "type": "bitmap",
   "size": 32,
   "default": {"or": [{"shift": [32768, 16]}, {"shift": [1, 9]}, {"shift": [0, 8]}, 4]}
  },

  {"name": "oxm_tlv_in_phy_port",
   "type": "record",
   "fields": [
     {"name": "header", "type": "oxm_tlv_header_in_phy_port"},
     {"name": "tlv", "type": "uint_32", "default": [0,0,0,0]}
   ]
  },

  {"name": "oxm_tlv_header_metadata",

```

```

"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[2, 9]}, {"shift":[0, 8]}, 8]}
},

{"name":"oxm_tlv_metadata",
"type":"record",
"fields":[
{"name":"header", "type":"oxm_tlv_header_metadata"},
{"name":"tlv", "type":"uint_32", "default":[0,0,0,0,0,0,0]}
]
},

{"name":"oxm_tlv_header_eth_dst",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[3, 9]}, {"shift":[0, 8]}, 6]}
},

{"name":"oxm_tlv_eth_dst",
"type":"record",
"fields":[
{"name":"header", "type":"oxm_tlv_header_eth_dst"},
{"name":"tlv", "type":"uint_48", "default":[0,0,0,0,0,0,0]}
]
},

{"name":"oxm_tlv_header_eth_src",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[4, 9]}, {"shift":[0, 8]}, 6]}
},

{"name":"oxm_tlv_eth_src",
"type":"record",
"fields":[
{"name":"header", "type":"oxm_tlv_header_eth_src"},
{"name":"tlv", "type":"uint_48", "default":[0,0,0,0,0,0,0]}
]
},

{"name":"oxm_tlv_header_eth_type",
"type":"bitmap",
"size":32,

```

```
"default":{"or":[{"shift":[32768, 16]}, {"shift":[5, 9]}, {"shift":[0, 8]}, 2]}
},

{"name":"oxm_tlv_eth_type",
"type":"record",
"fields":[
  {"name":"header", "type":"oxm_tlv_header_eth_type"},
  {"name":"tlv", "type":"uint_16", "default":[0,0]}
]
},

{"name":"oxm_tlv_header_vlan_vid",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[6, 9]}, {"shift":[0, 8]}, 2]}
},

{"name":"oxm_tlv_vlan_vid",
"type":"record",
"fields":[
  {"name":"header", "type":"oxm_tlv_header_vlan_vid"},
  {"name":"tlv", "type":"uint_16", "default":[0,0]}
]
},

{"name":"oxm_tlv_header_vlan_pcp",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[7, 9]}, {"shift":[0, 8]}, 1]}
},

{"name":"oxm_tlv_vlan_pcp",
"type":"record",
"fields":[
  {"name":"header", "type":"oxm_tlv_header_vlan_pcp"},
  {"name":"tlv", "type":"uint_16", "default":[0]}
]
},

{"name":"oxm_tlv_header_ip_dscp",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[8, 9]}, {"shift":[0, 8]}, 1]}
},
```

```
{ "name": "oxm_tlv_ip_dscp",
  "type": "record",
  "fields": [
    { "name": "header", "type": "oxm_tlv_header_ip_dscp" },
    { "name": "tlv", "type": "uint_16", "default": [0] }
  ]
},

{ "name": "oxm_tlv_header_ip_ecn",
  "type": "bitmap",
  "size": 32,
  "default": { "or": [ { "shift": [32768, 16] }, { "shift": [9, 9] }, { "shift": [0, 8] }, 1 ] }
},

{ "name": "oxm_tlv_ip_ecn",
  "type": "record",
  "fields": [
    { "name": "header", "type": "oxm_tlv_header_ip_ecn" },
    { "name": "tlv", "type": "uint_8", "default": [0] }
  ]
},

{ "name": "oxm_tlv_header_ip_proto",
  "type": "bitmap",
  "size": 32,
  "default": { "or": [ { "shift": [32768, 16] }, { "shift": [10, 9] }, { "shift": [0, 8] }, 1 ] }
},

{ "name": "oxm_tlv_ip_proto",
  "type": "record",
  "fields": [
    { "name": "header", "type": "oxm_tlv_header_ip_proto" },
    { "name": "tlv", "type": "uint_8", "default": [0] }
  ]
},

{ "name": "oxm_tlv_header_ipv4_src",
  "type": "bitmap",
  "size": 32,
  "default": { "or": [ { "shift": [32768, 16] }, { "shift": [11, 9] }, { "shift": [0, 8] }, 4 ] }
},

{ "name": "oxm_tlv_ipv4_src",
```

```
"type": "record",
"fields": [
  {"name": "header", "type": "oxm_tlv_header_ipv4_src"},
  {"name": "tlv", "type": "uint_32", "default": [0,0,0,0]}
]
},

{"name": "oxm_tlv_header_ipv4_dst",
"type": "bitmap",
"size": 32,
"default": {"or": [{"shift": [32768, 16]}, {"shift": [12, 9]}, {"shift": [0, 8]}, 4]}
},

{"name": "oxm_tlv_ipv4_dst",
"type": "record",
"fields": [
  {"name": "header", "type": "oxm_tlv_header_ipv4_dst"},
  {"name": "tlv", "type": "uint_32", "default": [0,0,0,0]}
]
},

{"name": "oxm_tlv_header_sctp_src",
"type": "bitmap",
"size": 32,
"default": {"or": [{"shift": [32768, 16]}, {"shift": [17, 9]}, {"shift": [0, 8]}, 2]}
},

{"name": "oxm_tlv_sctp_src",
"type": "record",
"fields": [
  {"name": "header", "type": "oxm_tlv_header_sctp_src"},
  {"name": "tlv", "type": "uint_16", "default": [0,0]}
]
},

{"name": "oxm_tlv_header_sctp_dst",
"type": "bitmap",
"size": 32,
"default": {"or": [{"shift": [32768, 16]}, {"shift": [18, 9]}, {"shift": [0, 8]}, 2]}
},

{"name": "oxm_tlv_sctp_dst",
```

```

"type":"record",
"fields":[
  {"name":"header", "type":"oxm_tlv_header_sctp_dst"},
  {"name":"tlv", "type":"uint_16", "default":[0,0]}
]
},

{"name":"oxm_tlv_header_icmpv4_type",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[19, 9]}, {"shift":[0, 8]}, 1]}
},

{"name":"oxm_tlv_icmpv4_type",
"type":"record",
"fields":[
  {"name":"header", "type":"oxm_tlv_header_icmpv4_type"},
  {"name":"tlv", "type":"uint_8", "default":[0]}
]
},

{"name":"oxm_tlv_header_icmpv4_code",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[20, 9]}, {"shift":[0, 8]}, 1]}
},

{"name":"oxm_tlv_icmpv4_code",
"type":"record",
"fields":[
  {"name":"header", "type":"oxm_tlv_header_icmpv4_code"},
  {"name":"tlv", "type":"uint_8", "default":[0]}
]
},

{"name":"oxm_tlv_header_arp_op",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[21, 9]}, {"shift":[0, 8]}, 2]}
},

{"name":"oxm_tlv_arp_op",
"type":"record",
"fields":[

```

```

    {"name":"header", "type":"oxm_tlv_header_arp_op"},
    {"name":"tlv", "type":"uint_16", "default":[0,0]}
  ]
},

{"name":"oxm_tlv_header_arp_spa",
 "type":"bitmap",
 "size":32,
 "default":{"or":[{"shift":[32768, 16]}, {"shift":[22, 9]}, {"shift":[0, 8]}, 4]}
},

{"name":"oxm_tlv_arp_spa",
 "type":"record",
 "fields":[
  {"name":"header", "type":"oxm_tlv_header_arp_spa"},
  {"name":"tlv", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"oxm_tlv_header_arp_tpa",
 "type":"bitmap",
 "size":32,
 "default":{"or":[{"shift":[32768, 16]}, {"shift":[23, 9]}, {"shift":[0, 8]}, 4]}
},

{"name":"oxm_tlv_arp_tpa",
 "type":"record",
 "fields":[
  {"name":"header", "type":"oxm_tlv_header_arp_tpa"},
  {"name":"tlv", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"oxm_tlv_header_arp_sha",
 "type":"bitmap",
 "size":32,
 "default":{"or":[{"shift":[32768, 16]}, {"shift":[24, 9]}, {"shift":[0, 8]}, 6]}
},

{"name":"oxm_tlv_arp_sha",
 "type":"record",
 "fields":[
  {"name":"header", "type":"oxm_tlv_header_arp_sha"},
  {"name":"tlv", "type":"uint_48", "default":[0,0,0,0,0,0]}
 ]
}

```

```

]
},

{"name":"oxm_tlv_header_arp_tha",
 "type":"bitmap",
 "size":32,
 "default":{"or":[{"shift":[32768, 16]}, {"shift":[25, 9]}, {"shift":[0, 8]}, 6]}
},

{"name":"oxm_tlv_arp_tha",
 "type":"record",
 "fields":[
  {"name":"header", "type":"oxm_tlv_header_arp_tha"},
  {"name":"tlv", "type":"uint_48", "default":[0,0,0,0,0,0]}
 ]
},

{"name":"oxm_tlv_header_ipv6_src",
 "type":"bitmap",
 "size":32,
 "default":{"or":[{"shift":[32768, 16]}, {"shift":[26, 9]}, {"shift":[0, 8]}, 16]}
},

{"name":"oxm_tlv_ipv6_src",
 "type":"record",
 "fields":[
  {"name":"header", "type":"oxm_tlv_header_ipv6_src"},
  {"name":"tlv", "type":"uint_128", "default":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]}
 ]
},

{"name":"oxm_tlv_header_ipv6_dst",
 "type":"bitmap",
 "size":32,
 "default":{"or":[{"shift":[32768, 16]}, {"shift":[27, 9]}, {"shift":[0, 8]}, 16]}
},

{"name":"oxm_tlv_ipv6_dst",
 "type":"record",
 "fields":[
  {"name":"header", "type":"oxm_tlv_header_ipv6_dst"},
  {"name":"tlv", "type":"uint_128", "default":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]}
 ]
},

```

```
{ "name": "oxm_tlv_header_ipv6_flabel",  
  "type": "bitmap",  
  "size": 32,  
  "default": { "or": [ { "shift": [32768, 16] }, { "shift": [28, 9] }, { "shift": [0, 8] }, 3 ] }  
},
```

```
{ "name": "oxm_tlv_ipv6_flabel",  
  "type": "record",  
  "fields": [  
    { "name": "header", "type": "oxm_tlv_header_ipv6_flabel" },  
    { "name": "tlv", "type": "uint_24", "default": [0,0,0] }  
  ]  
},
```

```
{ "name": "oxm_tlv_header_icmpv6_type",  
  "type": "bitmap",  
  "size": 32,  
  "default": { "or": [ { "shift": [32768, 16] }, { "shift": [29, 9] }, { "shift": [0, 8] }, 1 ] }  
},
```

```
{ "name": "oxm_tlv_icmpv6_type",  
  "type": "record",  
  "fields": [  
    { "name": "header", "type": "oxm_tlv_header_icmpv6_type" },  
    { "name": "tlv", "type": "uint_8", "default": [0] }  
  ]  
},
```

```
{ "name": "oxm_tlv_header_icmpv6_code",  
  "type": "bitmap",  
  "size": 32,  
  "default": { "or": [ { "shift": [32768, 16] }, { "shift": [30, 9] }, { "shift": [0, 8] }, 1 ] }  
},
```

```
{ "name": "oxm_tlv_icmpv6_code",  
  "type": "record",  
  "fields": [  
    { "name": "header", "type": "oxm_tlv_header_icmpv6_code" },  
    { "name": "tlv", "type": "uint_8", "default": [0] }  
  ]  
},
```

```
{ "name": "oxm_tlv_header_ipv6_nd_target",
```

```

"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[31, 9]}, {"shift":[0, 8]}, 16]}
},

{"name":"oxm_tlv_ipv6_nd_target",
"type":"record",
"fields":[
{"name":"header", "type":"oxm_tlv_header_ipv6_nd_target"},
{"name":"tlv", "type":"uint_128", "default":[0,0,0,0,0,0,0,0,0,0,0,0,0,0]}
]
},

{"name":"oxm_tlv_header_ipv6_nd_sll",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[32, 9]}, {"shift":[0, 8]}, 6]}
},

{"name":"oxm_tlv_ipv6_nd_sll",
"type":"record",
"fields":[
{"name":"header", "type":"oxm_tlv_header_ipv6_nd_sll"},
{"name":"tlv", "type":"uint_48", "default":[0,0,0,0,0,0]}
]
},

{"name":"oxm_tlv_header_ipv6_nd_tll",
"type":"bitmap",
"size":32,
"default":{"or":[{"shift":[32768, 16]}, {"shift":[33, 9]}, {"shift":[0, 8]}, 6]}
},

{"name":"oxm_tlv_ipv6_nd_tll",
"type":"record",
"fields":[
{"name":"header", "type":"oxm_tlv_header_ipv6_nd_tll"},
{"name":"tlv", "type":"uint_48", "default":[0,0,0,0,0,0]}
]
},

{"name":"oxm_tlv_header_ipv6_tc",
"type":"bitmap",
"size":32,

```

```
"default":{"or":[{"shift":[32768, 16]}, {"shift":[32, 9]}, {"shift":[0, 8]}, 6]}
},
```

```
{ "name":"oxm_tlv_ipv6_tc",
  "type":"record",
  "fields":[
    { "name":"header", "type":"oxm_tlv_header_ipv6_tc"},
    { "name":"tlv", "type":"uint_8", "default":[0]}
  ]
},
{ "name":"oxm_tlv_header_ipv6_pn",
  "type":"bitmap",
  "size":32,
  "default":{"or":[{"shift":[32768, 16]}, {"shift":[32, 9]}, {"shift":[0, 8]}, 6]}
},
```

```
{ "name":"oxm_tlv_ipv6_pn",
  "type":"record",
  "fields":[
    { "name":"header", "type":"oxm_tlv_header_ipv6_pn"},
    { "name":"tlv", "type":"uint_48", "default":[0,0,0,0,0,0]}
  ]
},
```

```
{ "name":"oxm_tlv_header_mpls_label",
  "type":"bitmap",
  "size":32,
  "default":{"or":[{"shift":[32768, 16]}, {"shift":[34, 9]}, {"shift":[0, 8]}, 3]}
},
```

```
{ "name":"oxm_tlv_mpls_label",
  "type":"record",
  "fields":[
    { "name":"header", "type":"oxm_tlv_header_mpls_label"},
    { "name":"tlv", "type":"uint_24", "default":[0,0,0]}
  ]
},
```

```
{ "name":"oxm_tlv_header_mpls_tc",
  "type":"bitmap",
  "size":32,
  "default":{"or":[{"shift":[32768, 16]}, {"shift":[35, 9]}, {"shift":[0, 8]}, 1]}
},
```

```
{ "name": "oxm_tlv_mpls_tc",  
  "type": "record",  
  "fields": [  
    { "name": "header", "type": "oxm_tlv_header_mpls_tc" },  
    { "name": "tlv", "type": "uint_8", "default": [0] }  
  ]  
},
```

```
{ "name": "oxm_tlv",  
  "type": "record",  
  "fields": [  
    { "name": "match", "type": ["oxm_tlv_ingress_port",  
      "oxm_tlv_in_phy_port",  
      "oxm_tlv_metadata",  
      "oxm_tlv_eth_dst",  
      "oxm_tlv_eth_src",  
      "oxm_tlv_eth_type",  
      "oxm_tlv_vlan_vid",  
      "oxm_tlv_vlan_pcp",  
      "oxm_tlv_ip_dscp",  
      "oxm_tlv_ip_ecn",  
      "oxm_tlv_ip_proto",  
      "oxm_tlv_ipv4_src",  
      "oxm_tlv_ipv4_dst",  
  
      "oxm_tlv_sctp_src",  
      "oxm_tlv_sctp_dst",  
      "oxm_tlv_icmpv4_type",  
      "oxm_tlv_icmpv4_code",  
      "oxm_tlv_arp_op",  
      "oxm_tlv_arp_spa",  
      "oxm_tlv_arp_tpa",  
      "oxm_tlv_arp_sha",  
      "oxm_tlv_arp_tha",  
      "oxm_tlv_ipv6_src",  
      "oxm_tlv_ipv6_dst",  
      "oxm_tlv_ipv6_flabel",  
      "oxm_tlv_icmpv6_type",  
      "oxm_tlv_icmpv6_code",
```

```

        "oxm_tlv_ipv6_nd_target",
        "oxm_tlv_ipv6_nd_sll",
        "oxm_tlv_ipv6_nd_tll",
        "oxm_tlv_mpls_label",
        "oxm_tlv_mpls_tc",

    ]}

    ],
    {
        "name": "match_header",
        "type": "record",
        "fields": [
            {
                "name": "type",
                "type": "ofp_match_type",
                "default": "OFPMT_OXM"
            },
            {
                "name": "length",
                "type": "uint_16",
                "default": [0,8]
            }
        ]
    },
    {
        "name": "oxm_tlv_fields",
        "type": "record",
        "fields": [
            {
                "name": "oxm_tlvs",
                "type": {
                    "type": "array",
                    "items": "oxm_tlv"
                }
            }
        ]
    },
    {
        "name": "ofp_match",
        "type": "record",
        "fields": [
            {
                "name": "header",
                "type": "match_header"
            },
            {
                "name": "fields",
                "type": "oxm_tlv_fields"
            },
            {
                "name": "closing_pad",
                "type": "bytes"
            }
        ]
    },
    {
        "name": "ofp_action_type",
        "type": "enum",
        "items": "uint_16",
        "list": [
            {
                "name": "OFPAT_OUTPUT",
                "default": [0,0]
            },
            {
                "name": "OFPAT_COPY_TTL_OUT",
                "default": [0,11]
            },
            {
                "name": "OFPAT_COPY_TTL_IN",
                "default": [0,12]
            },
            {
                "name": "OFPAT_SET_MPLS_TTL",
                "default": [0,15]
            },
            {
                "name": "OFPAT_DEC_MPLS_TTL",
                "default": [0,16]
            },
            {
                "name": "OFPAT_PUSH_VLAN",
                "default": [0,17]
            }
        ]
    }

```

```

    {"name":"OFPAT_POP_VLAN", "default":[0,18]},
    {"name":"OFPAT_PUSH_MPLS", "default":[0,19]},
    {"name":"OFPAT_POP_MPLS", "default":[0,20]},
    {"name":"OFPAT_SET_QUEUE", "default":[0,21]},
    {"name":"OFPAT_GROUP", "default":[0,22]},
    {"name":"OFPAT_SET_NW_TTL", "default":[0,23]},
    {"name":"OFPAT_DEC_NW_TTL", "default":[0,24]},
    {"name":"OFPAT_SET_FIELD", "default":[0,25]},
    {"name":"OFPAT_PUSH_PBB", "default":[0,26]},
    {"name":"OFPAT_POP_PBB", "default":[0,27]},
    {"name":"OFPIT_EXPERIMENTER", "default":[255,255]}]
},

{"name":"action_length",
 "type":"enum",
 "items":"uint_16",
 "list":[{"name":"AL_OUTPUT", "default":[0,16]},
        {"name":"AL_COPY_TTL_OUT", "default":[0,8]},
        {"name":"AL_COPY_TTL_IN", "default":[0,8]},
        {"name":"AL_SET_MPLS_TTL", "default":[0,8]},
        {"name":"AL_DEC_MPLS_TTL", "default":[0,8]},
        {"name":"AL_PUSH_VLAN", "default":[0,8]},
        {"name":"AL_POP_VLAN", "default":[0,8]},
        {"name":"AL_PUSH_MPLS", "default":[0,8]},
        {"name":"AL_POP_MPLS", "default":[0,8]},
        {"name":"AL_SET_QUEUE", "default":[0,8]},
        {"name":"AL_GROUP", "default":[0,8]},
        {"name":"AL_SET_NW_TTL", "default":[0,8]},
        {"name":"AL_DEC_NW_TTL", "default":[0,8]},
        {"name":"AL_SET_FIELD", "default":[0,8]},
        {"name":"AL_PUSH_PBB", "default":[0,8]},
        {"name":"AL_POP_PBB", "default":[0,8]},
        {"name":"AL_EXPERIMENTER", "default":[255,255]}]
},

{"name":"ofp_action_header",
 "type":"record",
 "fields":[
    {"name":"type", "type":"ofp_action_type"},
    {"name":"len", "type":"uint_16"},
    {"name":"pad", "type":"pad_4"}
 ]
},

```

```

{"name":"ofp_controller_max_len",
 "type":"enum",
 "items":"uint_16",
 "list":[{"name":"OFPCML_MAX", "default":[255, 229]},
        {"name":"OFPCML_NO_BUFFER", "default":[255, 255]}]
},

{"name":"ofp_action_output",
 "type":"record",
 "fields":[
  {"name":"type", "type":"ofp_action_type", "default":"OFPAT_OUTPUT"},
  {"name":"len", "type":"action_length", "default":"AL_OUTPUT"},
  {"name":"port", "type":"uint_32", "default":[0,0,0,0]},
  {"name":"max_len", "type":"ofp_controller_max_len",
   "default":"OFPCML_NO_BUFFER"},
  {"name":"pad", "type":"pad_6", "default":[0,0,0,0,0,0]}
 ]
},

{"name":"action_set_field_header",
 "type":"record",
 "fields":[
  {"name":"type", "type":"ofp_action_type", "default":"OFPAT_SET_FIELD"},
  {"name":"length", "type":"action_length", "default":"AL_SET_FIELD"}
 ]
},

{"name":"ofp_action_set_field",
 "type":"record",
 "fields":[
  {"name":"header", "type":"action_set_field_header"},
  {"name":"fields", "type":"oxm_tlv_fields"},
  {"name":"closing_pad", "type":"bytes"}
 ]
},

{"name":"action_pop_vlan_header",
 "type":"record",
 "fields":[
  {"name":"type", "type":"ofp_action_type", "default":"OFPAT_POP_VLAN"},
  {"name":"length", "type":"action_length", "default":"AL_POP_VLAN"}
 ]
},

```

```

{"name":"ofp_action_pop_vlan",
 "type":"record",
 "fields":[
  {"name":"header", "type":"action_pop_vlan_header"},
  {"name":"pad", "type":"pad_4"}
 ]
},

{"name":"action_push_mpls_header",
 "type":"record",
 "fields":[
  {"name":"type", "type":"ofp_action_type", "default":"OFPAT_PUSH_MPLS"},
  {"name":"length", "type":"action_length", "default":"AL_PUSH_MPLS"}
 ]
},

{"name":"ofp_action_push_mpls",
 "type":"record",
 "fields":[
  {"name":"header", "type":"action_push_mpls_header"},
  {"name":"ethertype", "type":"uint_16"},
  {"name":"pad", "type":"pad_2"}
 ]
},

{"name":"action_set_queue_header",
 "type":"record",
 "fields":[
  {"name":"type", "type":"ofp_action_type", "default":"OFPAT_SET_QUEUE"},
  {"name":"length", "type":"action_length", "default":"AL_SET_QUEUE"}
 ]
},

{"name":"ofp_action_set_queue",
 "type":"record",
 "fields":[
  {"name":"header", "type":"action_set_queue_header"},
  {"name":"queue_id", "type":"uint_32"}
 ]
},

{"name":"ofp_action",
 "type":"record",
 "fields":[

```

```

    {"name":"action",      "type":["ofp_action_output",      "ofp_action_set_field",
"ofp_action_pop_vlan", "ofp_action_push_mpls", "ofp_action_set_queue"]}
  ]
},

{"name":"action_set",
 "type":"record",
 "fields":[
  {"name":"set", "type":{"type":"array", "items":"ofp_action"}}
 ]
},

{"name":"ofp_instruction_type",
 "type":"enum",
 "items":"uint_16",
 "list":[{"name":"OFPIT_GOTO_TABLE", "default":[0,1]},
        {"name":"OFPIT_WRITE_METADATA", "default":[0,2]},
        {"name":"OFPIT_WRITE_ACTIONS", "default":[0,3]},
        {"name":"OFPIT_APPLY_ACTIONS", "default":[0,4]},
        {"name":"OFPIT_CLEAR_ACTIONS", "default":[0,5]},
        {"name":"OFPIT_EXPERIMENTER", "default":[255,255]}]
},

{"name":"instruction_length",
 "type":"enum",
 "items":"uint_16",
 "list":[{"name":"IL_GOTO_TABLE", "default":[0,8]},
        {"name":"IL_WRITE_METADATA", "default":[0,24]},
        {"name":"IL_WRITE_ACTIONS", "default":[0,8]},
        {"name":"IL_APPLY_ACTIONS", "default":[0,8]},
        {"name":"IL_CLEAR_ACTIONS", "default":[0,8]},
        {"name":"IL_EXPERIMENTER", "default":[255,255]}]
},

{"name":"instruction_goto_table_header",
 "type":"record",
 "fields":[
  {"name":"type",      "type":"ofp_instruction_type",
"default":"OFPIT_GOTO_TABLE"},
  {"name":"length", "type":"instruction_length", "default":"IL_GOTO_TABLE"}
 ]
},

{"name":"ofp_instruction_goto_table",

```

```

    "type": "record",
    "fields": [
      {"name": "header", "type": "instruction_goto_table_header"},
      {"name": "table_id", "type": "uint_8", "default": [0]},
      {"name": "pad", "type": "pad_3", "default": [0,0,0]}
    ]
  },

  {"name": "instruction_write_metadata_header",
   "type": "record",
   "fields": [
     {"name": "type", "type": "ofp_instruction_type",
      "default": "OFPIT_WRITE_METADATA"},
     {"name": "length", "type": "bytes"}
   ]
  },

  {"name": "instruction_write_actions_header",
   "type": "record",
   "fields": [
     {"name": "type", "type": "ofp_instruction_type",
      "default": "OFPIT_WRITE_ACTIONS"},
     {"name": "length", "type": "instruction_length",
      "default": "IL_WRITE_ACTIONS"},
     {"name": "pad", "type": "pad_4", "default": [0,0,0,0]}
   ]
  },

  {"name": "instruction_apply_actions_header",
   "type": "record",
   "fields": [
     {"name": "type", "type": "ofp_instruction_type",
      "default": "OFPIT_APPLY_ACTIONS"},
     {"name": "length", "type": "instruction_length",
      "default": "IL_APPLY_ACTIONS"},
     {"name": "pad", "type": "pad_4", "default": [0,0,0,0]}
   ]
  },

  {"name": "instruction_clear_actions_header",
   "type": "record",
   "fields": [
     {"name": "type", "type": "ofp_instruction_type",
      "default": "OFPIT_CLEAR_ACTIONS"},

```

```

        {"name":"length", "type":"instruction_length",
"default":"IL_CLEAR_ACTIONS"},
        {"name":"pad", "type":"pad_4", "default":[0,0,0,0]}
    ]
},

{"name":"ofp_instruction_actions",
"type":"record",
"fields":[
    {"name":"header", "type":["instruction_write_actions_header",
"instruction_apply_actions_header", "instruction_clear_actions_header"]},
    {"name":"actions", "type":"action_set"}
]
},

{"name":"ofp_instruction_apply_actions",
"type":"record",
"fields":[
    {"name":"header", "type":"instruction_apply_actions_header"},
    {"name":"actions", "type":"action_set"}
]
},

{"name":"ofp_instruction_write_metadata",
"type":"record",
"fields":[
    {"name":"header", "type":"instruction_write_metadata_header"},
    {"name":"table_id", "type":"bytes"},
    {"name":"pad", "type":"pad_4"}
]
},

{"name":"ofp_instruction_write_actions",
"type":"record",
"fields":[
    {"name":"header", "type":"instruction_write_actions_header"},
    {"name":"table_id", "type":"bytes"},
    {"name":"pad", "type":"pad_4"}
]
},

{"name":"ofp_instruction_clear_actions",
"type":"record",
"fields":[

```

```

    {"name":"header", "type":"instruction_clear_actions_header"},
    {"name":"table_id", "type":"bytes"},
    {"name":"pad", "type":"pad_4"}
  ]
},

{"name":"ofp_instruction",
 "type":"record",
 "fields":[
  {"name":"instruction", "type":["ofp_instruction_goto_table",
                                "ofp_instruction_write_metadata",
                                "ofp_instruction_write_actions",
                                "ofp_instruction_clear_actions",
                                "ofp_instruction_apply_actions"]}
 ]
},

{"name":"instruction_set",
 "type":"record",
 "fields":[
  {"name":"set", "type":{"type":"array", "items":["ofp_instruction", "null"]}}
 ]
},

{"name":"ofp_flow_mod_flags",
 "type":"enum",
 "items":"uint_16",
 "list":[{"name":"OFPFF_SEND_FLOW_REM", "default":[0,0]},
        {"name":"OFPFF_CHECK_OVERLAP", "default":[0,1]},
        {"name":"OFPFF_RESET_COUNTS", "default":[0,2]}]
},

{"name":"ofp_flow_mod_command",
 "type":"enum",
 "items":"uint_8",
 "list":[
  {"name":"OFPFC_ADD", "default":[0]},
  {"name":"OFPFC_MODIFY", "default":[1]},
  {"name":"OFPFC_MODIFY_STRICT", "default":[2]},
  {"name":"OFPFC_DELETE", "default":[3]},
  {"name":"OFPFC_DELETE_STRICT", "default":[4]}
 ]
},

```

```

{"name":"flow_mod_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_FLOW_MOD"},
  {"name":"length", "type":"ofp_length", "default":"OFPL_FLOW_MOD_LEN"},
  {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"flow_mod_body_add",
 "type":"record",
 "fields":[

  {"name":"table_id", "type":"uint_8", "default":[0]},
  {"name":"command", "type":"ofp_flow_mod_command",
"default":"OFPPFC_ADD"},
  {"name":"idle_timeout", "type":"uint_16", "default":[0,0]},
  {"name":"hard_timeout", "type":"uint_16", "default":[0,0]},
  {"name":"priority", "type":"uint_16", "default":[0,0]},
  {"name":"buffer_id", "type":"uint_32", "default":[0,0,0,0]},
  {"name":"out_port", "type":"uint_32", "default":[0,0,0,0]},
  {"name":"out_group", "type":"uint_32", "default":[0,0,0,0]},
  {"name":"flags", "type":"ofp_flow_mod_flags",
"default":"OFPPFF_SEND_FLOW_REM"},
  {"name":"pad", "type":"pad_2", "default":[0,0]}
 ]
},

{"name":"flow_mod_body_delete",
 "type":"record",
 "fields":[
  {"name":"cookie", "type":"uint_64", "default":[0,0,0,0,0,0,0,0]},
  {"name":"cookie_mask", "type":"uint_64", "default":[0,0,0,0,0,0,0,0]},
  {"name":"table_id", "type":"uint_8", "default":[0]},
  {"name":"command", "type":"ofp_flow_mod_command",
"default":"OFPPFC_DELETE"},
  {"name":"idle_timeout", "type":"uint_16", "default":[0,0]},
  {"name":"hard_timeout", "type":"uint_16", "default":[0,0]},
  {"name":"priority", "type":"uint_16", "default":[0,0]},
  {"name":"buffer_id", "type":"uint_32", "default":[0,0,0,0]},
  {"name":"out_port", "type":"uint_32", "default":[255,255,255,255]},
  {"name":"out_group", "type":"uint_32", "default":[255,255,255,255]},

```

```

        {"name":"flags",
"default":"OFPPF_SEND_FLOW_REM"},
        {"name":"pad", "type":"pad_2", "default":[0,0]}
    ]
},

{"name":"ofp_flow_mod",
"type":"record",
"fields":[
    {"name":"header", "type":"flow_mod_header"},
    {"name":"base", "type":["flow_mod_body_add","flow_mod_body_delete"]},
    {"name":"match", "type":"ofp_match"},
    {"name":"instructions", "type":"instruction_set"}
]
},

{"name":"group_mod_header",
"type":"record",
"fields":[
    {"name":"version", "type":"uint_8", "default":[4]},
    {"name":"type", "type":"ofp_type", "default":"OFPT_GROUP_MOD"},
    {"name":"length", "type":"ofp_length", "default":"OFPL_GROUP_MOD_LEN"},
    {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
]
},

{"name":"bucket_body",
"type":"record",
"fields":[
    {"name":"len", "type":"uint_16", "default":[0,16]},
    {"name":"weight", "type":"uint_16", "default":[0,0]},
    {"name":"watch_port", "type":"uint_32", "default":[0,0,0,0]},
    {"name":"watch_group", "type":"uint_32", "default":[0,0,0,0]},
    {"name":"pad", "type":"pad_4", "default":[0,0,0,0]}
]
},

{"name":"ofp_bucket",
"type":"record",
"fields":[
    {"name":"body", "type":"bucket_body"},
    {"name":"actions", "type":["action_set", "null"]}
]
},

```

```

{"name":"bucket_set",
 "type":"record",
 "fields":[
  {"name":"set", "type":{"type":"array", "items":"ofp_bucket"}}
 ]
},

{"name":"ofp_group_mod_command",
 "type":"enum",
 "items":"uint_16",
 "list":[
  {"name":"OFPGC_ADD", "default":[0,0]},
  {"name":"OFPGC_MODIFY", "default":[0,1]},
  {"name":"OFPGC_DELETE", "default":[0,2]}
 ]
},

{"name":"ofp_group_type",
 "type":"enum",
 "items":"uint_8",
 "list":[
  {"name":"OFPGT_ALL", "default":[0,0]},
  {"name":"OFPGT_SELECT", "default":[0,1]},
  {"name":"OFPGT_INDIRECT", "default":[0,2]},
  {"name":"OFPGT_FF", "default":[0,3]}
 ]
},

{"name":"ofp_group",
 "type":"enum",
 "items":"uint_32",
 "list":[
  {"name":"OFPG_MAX", "default":[255,255,255,0]},
  {"name":"OFPG_ALL", "default":[255,255,255,252]},
  {"name":"OFPG_ANY", "default":[255,255,255,255]}
 ]
},

{"name":"group_mod_body",
 "type":"record",
 "fields":[
  {"name":"command",
"default":"OFPGC_ADD"},
"ofp_group_mod_command",

```

```

    {"name":"type", "type":"ofp_group_type", "default":"OFPGT_ALL"},
    {"name":"pad", "type":"pad_1", "default":[0]},
    {"name":"group_id", "type":"ofp_group", "default":"OFPG_ANY"}
  ]
},

{"name":"ofp_group_mod",
 "type":"record",
 "fields":[
  {"name":"header", "type":"group_mod_header"},
  {"name":"body", "type":"group_mod_body"},
  {"name":"buckets", "type":"bucket_set"}
 ]
},

{"name":"port_mod_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_PORT_MOD"},
  {"name":"length", "type":"ofp_length", "default":"OFPL_FLOW_MOD_LEN"},
  {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"OFPPC",
 "type":"record",
 "fields":[]
},

{"name":"ofp_port_mod",
 "type":"record",
 "fields":[
  {"name":"header", "type":"port_mod_header"},
  {"name":"port_no", "type":"uint_32"},
  {"name":"pad", "type":"pad_4"},

  {"name":"pad2", "type":"pad_2"},
  {"name":"config", "type":"OFPPC"},
  {"name":"mask", "type":"OFPPC"},
  {"name":"advertise", "type":"OFPPC"},
  {"name":"pad3", "type":"pad_3"}
 ]
}

```

```

]
},

{"name":"echo_request_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_ECHO_REQUEST"},
  {"name":"length", "type":"ofp_length",
"default":"OFPL_ECHO_REQUEST_LEN"},
  {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"ofp_echo_request",
 "type":"record",
 "fields":[
  {"name":"header", "type":"echo_request_header"}
 ]
},

{"name":"echo_reply_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_ECHO_REPLY"},
  {"name":"length", "type":"ofp_length",
"default":"OFPL_ECHO_REPLY_LEN"},
  {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"ofp_echo_reply",
 "type":"record",
 "fields":[
  {"name":"header", "type":"echo_reply_header"}
 ]
},

{"name":"ofp_packet_in_reason",
 "type":"enum",
 "items":"uint_8",
 "list": [

```

```

    {"name":"OFPR_NO_MATCH", "default":[0]},
    {"name":"OFPR_ACTION", "default":[1]},
    {"name":"OFPR_INVALID_TTL", "default":[2]}
  ]
},

{"name":"packet_in_header",
 "type":"record",
 "fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_PACKET_IN"},
  {"name":"length", "type":"ofp_length", "default":"OFPL_PACKET_IN_LEN"},
  {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
 ]
},

{"name":"ofp_packet_in",
 "type":"record",
 "fields":[
  {"name":"header", "type":"packet_in_header"},
  {"name":"buffer_id", "type":"uint_32"},
  {"name":"total_len", "type":"uint_16"},
  {"name":"reason", "type":"ofp_packet_in_reason"},
  {"name":"table_id", "type":"uint_8"},
  {"name":"cookie", "type":"uint_64"},
  {"name":"match", "type":"ofp_match"},
  {"name":"pad", "type":"pad_2"},
  {"name":"data", "type":{"type":"array", "items":"uint_8"}}
 ]
},

{"name":"ofp_port_reason",
 "type":"enum",
 "items":"uint_8",
 "list":[
  {"name":"OFPPR_ADD", "default":[0]},
  {"name":"OFPPR_DELETE", "default":[1]},
  {"name":"OFPPR_MODIFY", "default":[2]}
 ]
},

{"name":"port_status_header",
 "type":"record",

```

```
"fields":[
  {"name":"version", "type":"uint_8", "default":[4]},
  {"name":"type", "type":"ofp_type", "default":"OFPT_PORT_STATUS"},
  {"name":"length", "type":"ofp_length",
"default":"OFPL_PORT_STATUS_LEN"},
  {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
]
```

```
{ "name":"ofp_port_status",
  "type":"record",
  "fields":[
    {"name":"header", "type":"port_status_header"},
    {"name":"reason", "type":"ofp_port_reason"},
    {"name":"pad", "type":"pad_7"}
  ]
},
```

```
{ "name":"ofp_error_type",
  "type":"enum",
  "items":"uint_16",
  "list":[
    {"name":"OFPET_HELLO_FAILED", "default":[0,0]},
    {"name":"OFPET_BAD_REQUEST", "default":[0,1]},
    {"name":"OFPET_BAD_ACTION", "default":[0,2]},
    {"name":"OFPET_BAD_INSTRUCTION", "default":[0,3]},
    {"name":"OFPET_BAD_MATCH", "default":[0,4]},
    {"name":"OFPET_FLOW_MOD_FAILED", "default":[0,5]},
    {"name":"OFPET_GROUP_MOD_FAILED", "default":[0,6]},
    {"name":"OFPET_PORT_MOD_FAILED", "default":[0,7]},
    {"name":"OFPET_TABLE_MOD_FAILED", "default":[0,8]},
    {"name":"OFPET_QUEUE_OP_FAILED", "default":[0,9]},
    {"name":"OFPET_SWITCH_CONFIG_FAILED", "default":[0,10]},
    {"name":"OFPET_ROLE_REQUEST_FAILED", "default":[0,11]},
    {"name":"OFPET_METER_MOD_FAILED", "default":[0,12]},
    {"name":"OFPET_TABLE_FEATURES_FAILED", "default":[0,13]},
    {"name":"OFPET_EXPERIMENTER", "default":[255,255]}
  ]
},
```

```
{ "name":"ofp_hello_failed_code",
  "type":"enum",
  "items":"uint_16",
  "list": [
```

```

    {"name":"OFPHFC_INCOMPATIBLE", "default":[0,0]},
    {"name":"OFPHFC_EPERM", "default":[0,1]}
  ]
},

{"name":"ofp_bad_request_code",
 "type":"enum",
 "items":"uint_16",
 "list": [
  {"name":"OFPBRC_BAD_VERSION", "default":[0,0]},
  {"name":"OFPBRC_BAD_TYPE", "default":[0,1]},
  {"name":"OFPBRC_BAD_MULTIPART", "default":[0,2]},
  {"name":"OFPBRC_BAD_EXPERIMENTER", "default":[0,3]},
  {"name":"OFPBRC_BAD_EXP_TYPE", "default":[0,4]},
  {"name":"OFPBRC_EPERM", "default":[0,5]},
  {"name":"OFPBRC_BAD_LEN", "default":[0,6]},
  {"name":"OFPBRC_BUFFER_EMPTY", "default":[0,7]},
  {"name":"OFPBRC_BUFFER_UNKNOWN", "default":[0,8]},
  {"name":"OFPBRC_BAD_TABLE_ID", "default":[0,9]},
  {"name":"OFPBRC_IS_SLAVE", "default":[0,10]},
  {"name":"OFPBRC_BAD_PORT", "default":[0,11]},
  {"name":"OFPBRC_BAD_PACKET", "default":[0,12]},
  {"name":"OFPBRC_MULTIPART_BUFFER_OVERFLOW", "default":[0,13]}
  ]
},

{"name":"ofp_bad_action_code",
 "type":"enum",
 "items":"uint_16",
 "list": [
  {"name":"OFPBAC_BAD_TYPE", "default":[0,0]},
  {"name":"OFPBAC_BAD_LEN", "default":[0,1]},
  {"name":"OFPBAC_BAD_EXPERIMENTER", "default":[0,2]},
  {"name":"OFPBAC_BAD_EXP_TYPE", "default":[0,3]},
  {"name":"OFPBAC_BAD_OUT_PORT", "default":[0,4]},
  {"name":"OFPBAC_BAD_ARGUMENT", "default":[0,5]},
  {"name":"OFPBAC_EPERM", "default":[0,6]},
  {"name":"OFPBAC_TOO_MANY", "default":[0,7]},
  {"name":"OFPBAC_BAD_QUEUE", "default":[0,8]},
  {"name":"OFPBAC_BAD_OUT_GROUP", "default":[0,9]},
  {"name":"OFPBAC_MATCH_INCONSISTENT", "default":[0,10]},
  {"name":"OFPBAC_UNSUPPORTED_ORDER", "default":[0,11]},
  {"name":"OFPBAC_BAD_TAG", "default":[0,12]},
  {"name":"OFPBAC_BAD_SET_TYPE", "default":[0,13]},

```

```

    {"name":"OFFBAC_BAD_SET_LEN", "default":[0,14]},
    {"name":"OFFBAC_BAD_SET_ARGUMENT", "default":[0,15]}
  ]
},

{"name":"ofp_bad_instruction_code",
 "type":"enum",
 "items":"uint_16",
 "list": [
  {"name":"OFFBIC_UNKNOWN_INST", "default":[0,0]},
  {"name":"OFFBIC_UNSUP_INST", "default":[0,1]},
  {"name":"OFFBIC_BAD_TABLE_ID", "default":[0,2]},
  {"name":"OFFBIC_UNSUP_METADATA", "default":[0,3]},
  {"name":"OFFBIC_UNSUP_METADATA_MASK", "default":[0,4]},
  {"name":"OFFBIC_BAD_EXPERIMENTER", "default":[0,5]},
  {"name":"OFFBIC_BAD_EXP_TYPE", "default":[0,6]},
  {"name":"OFFBIC_BAD_LEN", "default":[0,7]},
  {"name":"OFFBIC_EPERM", "default":[0,8]}
 ]
},

{"name":"ofp_bad_match_code",
 "type":"enum",
 "items":"uint_16",
 "list": [
  {"name":"OFFBMC_BAD_TYPE", "default":[0,0]},
  {"name":"OFFBMC_BAD_LEN", "default":[0,1]},
  {"name":"OFFBMC_BAD_TAG", "default":[0,2]},
  {"name":"OFFBMC_BAD_DL_ADDR_MASK", "default":[0,3]},
  {"name":"OFFBMC_BAD_NW_ADDR_MASK", "default":[0,4]},
  {"name":"OFFBMC_BAD_WILDCARDS", "default":[0,5]},
  {"name":"OFFBMC_BAD_FIELD", "default":[0,6]},
  {"name":"OFFBMC_BAD_VALUE", "default":[0,7]},
  {"name":"OFFBMC_BAD_MASK", "default":[0,8]},
  {"name":"OFFBMC_BAD_PREREQ", "default":[0,9]},
  {"name":"OFFBMC_DUP_FIELD", "default":[0,10]},
  {"name":"OFFBMC_EPERM", "default":[0,11]}
 ]
},

{"name":"ofp_flow_mod_failed_code",
 "type":"enum",
 "items":"uint_16",
 "list": [

```

```

    {"name":"OFPFMFC_UNKNOWN", "default":[0,0]},
    {"name":"OFPFMFC_TABLE_FULL", "default":[0,1]},
    {"name":"OFPFMFC_BAD_TABLE_ID", "default":[0,2]},
    {"name":"OFPFMFC_OVERLAP", "default":[0,3]},
    {"name":"OFPFMFC_EPERM", "default":[0,4]},
    {"name":"OFPFMFC_BAD_TIMEOUT", "default":[0,5]},
    {"name":"OFPFMFC_BAD_COMMAND", "default":[0,6]},
    {"name":"OFPFMFC_BAD_FLAGS", "default":[0,7]}
  ]
},

{"name":"ofp_group_mod_failed_code",
 "type":"enum",
 "items":"uint_16",
 "list": [
  {"name":"OFPGMFC_GROUP_EXISTS", "default":[0,0]},
  {"name":"OFPGMFC_INVALID_GROUP", "default":[0,1]},
  {"name":"OFPGMFC_WEIGHT_UNSUPPORTED", "default":[0,2]},
  {"name":"OFPGMFC_OUT_OF_GROUPS", "default":[0,3]},
  {"name":"OFPGMFC_OUT_OF_BUCKETS", "default":[0,4]},
  {"name":"OFPGMFC_CHAINING_UNSUPPORTED", "default":[0,5]},
  {"name":"OFPGMFC_WATCH_UNSUPPORTED", "default":[0,6]},
  {"name":"OFPGMFC_LOOP", "default":[0,7]},
  {"name":"OFPGMFC_UNKNOWN_GROUP", "default":[0,8]},
  {"name":"OFPGMFC_CHAINED_GROUP", "default":[0,9]},
  {"name":"OFPGMFC_BAD_TYPE", "default":[0,10]},
  {"name":"OFPGMFC_BAD_COMMAND", "default":[0,11]},
  {"name":"OFPGMFC_BAD_BUCKET", "default":[0,12]},
  {"name":"OFPGMFC_BAD_WATCH", "default":[0,13]},
  {"name":"OFPGMFC_EPERM", "default":[0,14]}
  ]
},

{"name":"ofp_port_mod_failed_code",
 "type":"enum",
 "items":"uint_16",
 "list": [
  {"name":"OFPPMFC_BAD_PORT", "default":[0,0]},
  {"name":"OFPPMFC_BAD_HW_ADDR", "default":[0,1]},
  {"name":"OFPPMFC_BAD_CONFIG", "default":[0,2]},
  {"name":"OFPPMFC_BAD_ADVERTISE", "default":[0,3]},
  {"name":"OFPPMFC_EPERM", "default":[0,4]}
  ]
},

```

```
{ "name": "ofp_table_mod_failed_code",  
  "type": "enum",  
  "items": "uint_16",  
  "list": [  
    { "name": "OFPTMFC_BAD_TABLE", "default": [0,0] },  
    { "name": "OFPTMFC_BAD_CONFIG", "default": [0,1] },  
    { "name": "OFPTMFC_EPERM", "default": [0,2] }  
  ]  
},
```

```
{ "name": "ofp_queue_op_failed_code",  
  "type": "enum",  
  "items": "uint_16",  
  "list": [  
    { "name": "OFFQOFC_BAD_PORT", "default": [0,0] },  
    { "name": "OFFQOFC_BAD_QUEUE", "default": [0,1] },  
    { "name": "OFFQOFC_EPERM", "default": [0,2] }  
  ]  
},
```

```
{ "name": "ofp_switch_config_failed_code",  
  "type": "enum",  
  "items": "uint_16",  
  "list": [  
    { "name": "OFFSCFC_BAD_FLAGS", "default": [0,0] },  
    { "name": "OFFSCFC_BAD_LEN", "default": [0,1] },  
    { "name": "OFFSCFC_EPERM", "default": [0,2] }  
  ]  
},
```

```
{ "name": "ofp_role_request_failed_code",  
  "type": "enum",  
  "items": "uint_16",  
  "list": [  
    { "name": "OFFRRFC_STALE", "default": [0,0] },  
    { "name": "OFFRRFC_UNSUP", "default": [0,1] },  
    { "name": "OFFRRFC_BAD_ROLE", "default": [0,2] }  
  ]  
},
```

```
{ "name": "ofp_meter_mod_failed_code",  
  "type": "enum",  
  "items": "uint_16",
```

```
"list": [  
  {"name":"OFPMFC_UNKNOWN", "default":[0,0]},  
  {"name":"OFPMFC_METER_EXISTS", "default":[0,1]},  
  {"name":"OFPMFC_INVALID_METER", "default":[0,2]},  
  {"name":"OFPMFC_UNKNOWN_METER", "default":[0,3]},  
  {"name":"OFPMFC_BAD_COMMAND", "default":[0,4]},  
  {"name":"OFPMFC_BAD_FLAGS", "default":[0,5]},  
  {"name":"OFPMFC_BAD_RATE", "default":[0,6]},  
  {"name":"OFPMFC_BAD_BURST", "default":[0,7]},  
  {"name":"OFPMFC_BAD_BAND", "default":[0,8]},  
  {"name":"OFPMFC_BAD_BAND_VALUE", "default":[0,9]},  
  {"name":"OFPMFC_OUT_OF_METERS", "default":[0,10]},  
  {"name":"OFPMFC_OUT_OF_BANDS", "default":[0,11]}  
]  
},
```

```
{ "name":"ofp_table_features_failed_code",  
  "type":"enum",  
  "items":"uint_16",  
  "list": [  
    {"name":"OFPTFFC_BAD_TABLE", "default":[0,0]},  
    {"name":"OFPTFFC_BAD_METADATA", "default":[0,1]},  
    {"name":"OFPTFFC_BAD_TYPE", "default":[0,2]},  
    {"name":"OFPTFFC_BAD_LEN", "default":[0,3]},  
    {"name":"OFPTFFC_BAD_ARGUMENT", "default":[0,4]},  
    {"name":"OFPTFFC_EPERM", "default":[0,5]}  
  ]  
},
```

```
{ "name":"ofp_failed_code",  
  "type":"record",  
  "fields": [  
    {"name":"code", "type":["ofp_hello_failed_code",  
      "ofp_bad_request_code",  
      "ofp_bad_action_code",  
      "ofp_bad_instruction_code",  
      "ofp_bad_match_code",  
      "ofp_flow_mod_failed_code",  
      "ofp_group_mod_failed_code",  
      "ofp_port_mod_failed_code",  
      "ofp_table_mod_failed_code",  
      "ofp_queue_op_failed_code",  
      "ofp_switch_config_failed_code",  
      "ofp_role_request_failed_code",
```

```
        "ofp_meter_mod_failed_code",  
        "ofp_table_features_failed_code"  
    ]  
}  
]  
}
```

Appendix: B – Credits

Dr. Deniz Gurkan, Mahi Panahi, Sandhya Narayan and Dmitry Orekhov.