

Extensions to OpenFlow Protocol in support Circuit Switching

Addendum to OpenFlow Protocol Specification (v0.8.9) – Circuit Switch Addendum v0.2

August 22nd, 2009

Current Maintainer: Saurav Das (sd2@stanford.edu)

1. Introduction

This document describes the requirements of an OpenFlow circuit switch. We recommend that you read the version v0.8.9 of the OpenFlow switch specification for packet switches on the OpenFlow Consortium website (<http://OpenFlowSwitch.org>). This specification covers the components and basic functions of circuit switches based on switching time-slots, wavelengths and fibers. It also covers hybrid switches such as packet switches with circuit interfaces and conversely circuit switches with packet interfaces. Accordingly this document specifies OpenFlow protocol changes required to manage such OpenFlow switches from a remote controller. This document should be viewed as an addendum to the switch specification for packet switches and not independently.

2. Switch Components

An OpenFlow circuit switch consists of a cross-connect table, which caches the information about the existing circuit flows (or cross-connects made in the switch), and a secure channel to an external controller, which manages the switch over the secure channel using the OpenFlow protocol.

The circuit switch flow table (cross-connect table) contains a set of circuit flow entries, which detail which input channels are cross-connected to which output channels (Fig. 1). Additionally it maintains 1 or more actions for each circuit flow, details of which will be explained in following sections. Lastly, where appropriate it maintains statistics for the flow.

Circuit Flows

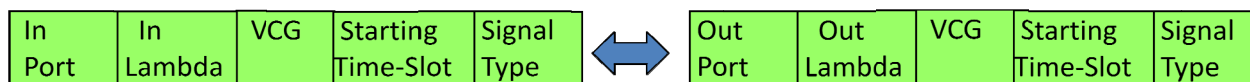


Fig. 1: Circuit flow table (cross-connect table) entry

Unlike an OpenFlow packet switch, the cross-connect table is not used to lookup flows, as circuits ports typically have no visibility into packets (the payload). Typically there is no buffering within a circuit switch (optical or electronic). Accordingly no packets are forwarded to the controller as well. The controller is responsible for provisioning and removing connections in an OpenFlow circuit switch via the extensions to the OpenFlow protocol for supporting circuit switching. Some modern circuit switches include packet interfaces and can switch packets electronically, or send them out of the circuit interfaces. Such switches should maintain separate packet and circuit flow tables as shown in Fig. 2a.

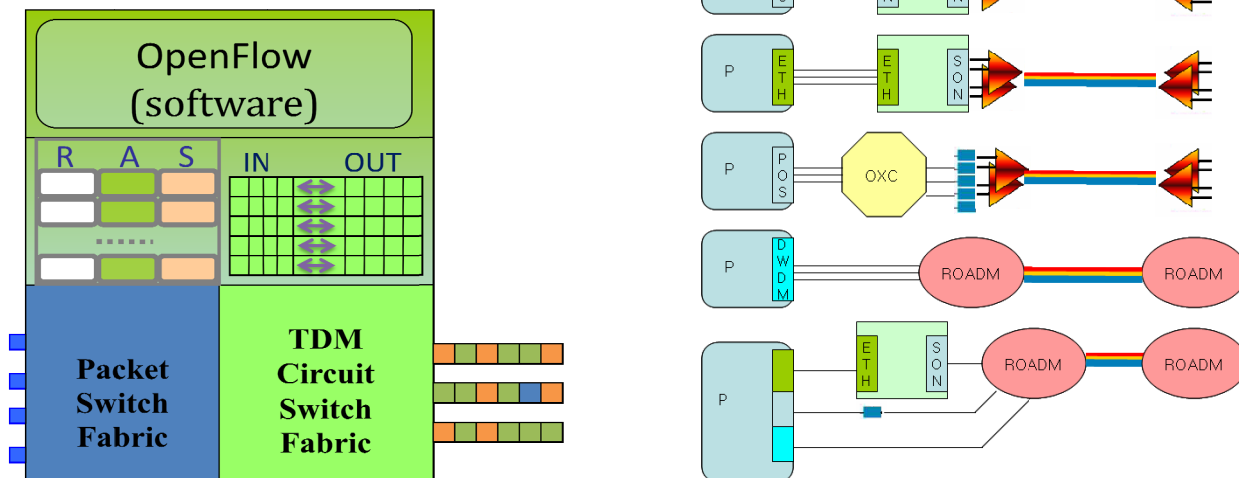


Fig 2(a) An OpenFlow switch with packet and circuit hardware flow tables; 2(b) Various ways to interconnect packet and circuit switches

Fig. 2b shows various ways to interconnect packet and circuit switches:

1. The packet switch P is connected to the TDM circuit switch C via a Packet over SONET (POS) interface. The SONET switches (DXC) are connected to each other over static DWDM line systems.
2. The same as choice 1, but now the P is connected to a TDM C via an Ethernet interface. C has the capability to adapt IP packets from the Ethernet interface to SONET framing.
3. P is now connected via an OXC such a fiber crossconnect. POS framing is used on the packet switch. This interface on the packet switch does not use high quality transceivers needed for long distance communications neither does it use the standardized ITU grid wavelengths– hence DWDM transponders are needed before the signal is transmitted over the DWDM line systems.
4. The packet switch now uses DWDM transceivers (with suitable framing). The wavelength on the transmitters may even be tunable. The static DWDM line system has now been replaced with more modern ROADM/WSS based OXCs.
5. Lastly, this configuration shows that the actual situation could be a combination of the above. Some interfaces on the packet switch could be connected to TDM circuit switches which themselves connect via ROADMs. Other interfaces can directly connect to the OXC via transponders or tunable DWDM interfaces.

Definition of terms:

- TDM/ DWDM – Time Division Multiplexing/ Dense Wavelength Division Multiplexing;
- SONET/SDH – Synchronous Optical NETWORK/ Synchronous Digital Hierarchy;
- DXC/OXC – Digital Cross-connect/ Optical Cross-connect;
- ROADM/WSS – Reconfigurable Optical Add Drop Multiplexer/ Wavelength Selective Switch;
- VCAT/VCG/LCAS – Virtual Concatenation/ Virtual Concatenation Group/ Link Capacity Adjustment Scheme
- POS/GFP – Packet over SONET framing/ Generic Framing Procedure
- ITU – International Telecommunications Union

3. OpenFlow Circuit-Switch Protocol

The OpenFlow protocol supports three message types, controller-to-switch, asynchronous and symmetric each with multiple subtypes. In general, we maintain the same message types for a circuit switch but change some of the structures used in the message. As mentioned in the introduction, this spec *should not* be viewed as independent of the packet switching spec. This spec extends structs and messages used in the packet spec. We first summarize the changes we have made to the packet switching spec v0.8.9 and then give details in subsequent sections.

3.1 Changes to OpenFlow packet switching spec v0.8.9

1. Changes to the capabilities field in struct `ofp_switch_features` to account for circuit switch capabilities not found in regular packet switches (Sec. 3.2).
2. Extension of struct `ofp_phy_port` to account for circuit port characteristics (Sec. 3.3)
3. Definition of struct `ofp_connect` for specifying the cross-connection in circuit switches. This struct is the logical equivalent of struct `ofp_match` in the packet spec (Sec. 3.4)
4. OpenFlow messages: this addendum defines one additional message to enum `ofp_type` – `OFPT_CFLOW_MOD` and one additional flow mod command `OFPMC_DROP` (Sec. 3.5)
5. Addition of two action types `OFPAT_CKT_OUTPUT` and `OFPAT_CKT_INPUT` to account for adapting packet flows to circuit flows and extracting packet flows from circuit flows (Sec. 3.6)
6. Addition of error messages to inform controller of problems in circuit switch configuration (Sec. 3.7)

3.2 Switch Features

Upon SSL session establishment, the controller sends an `OFPT_FEATURES_REQUEST` message. This message does not contain a body beyond the OpenFlow header. The switch responds with an `OFPT_FEATURES_REPLY` message:

```
/* Switch features. */
struct ofp_switch_features {
    struct ofp_header header;
    uint64_t datapath_id;          /* Datapath unique ID. Only the lower 48-bits are meaningful. */

    uint32_t n_buffers;           /* Max packets buffered at once. */
    uint8_t n_tables;            /* Number of tables supported by datapath. */
    uint8_t pad[3];              /* Align to 64-bits. */

    /* Features. */
    uint32_t capabilities;       /* Bitmap of supported "ofp_capabilities". */
    uint32_t actions;           /* Bitmap of supported "ofp_action_type"s. */

    /* Port info. */
    struct ofp_phy_port ports[0]; /* Port definitions. The number of ports is inferred from
                                   the length field in the header. */
};
OFP_ASSERT(sizeof(struct ofp_switch_features) == 32);
```

For an OpenFlow circuit switch, we maintain the same switch_features struct used for OF packet switches. However we expand the capabilities field to include the following:

```

/* Capabilities supported by the datapath. */
enum ofp_capabilities {
    OFPC_FLOW_STATS      = 1 << 0, /* Flow statistics. */
    OFPC_TABLE_STATS     = 1 << 1, /* Table statistics. */
    OFPC_PORT_STATS      = 1 << 2, /* Port statistics. */
    OFPC_STP             = 1 << 3, /* 802.1d spanning tree. */
    OFPC_MULTI_PHY_TX    = 1 << 4, /* Supports transmitting through multiple physical interfaces */
    OFPC_IP_REASM        = 1 << 5, /* Can reassemble IP fragments. */

    /* following capabilities are defined for circuit switches*/
    OFPC_CTG_CONCAT      = 1 << 31, /* Support for contiguous concatenation on all TDM ports */
    OFPC_VIR_CONCAT     = 1 << 30, /* Support for virtual concatenation (VCAT) on TDM ports*/
    OFPC_LCAS           = 1 << 29, /* Support for Link Capacity Adjustment Scheme (LCAS) */
    OFPC_POS            = 1 << 28, /* Support for Packet over Sonet (PoS) adaptation */
    OFPC_GFP            = 1 << 27, /* Support for Generic Framing Procedure (GFP) adaptation */
    OFPC_10G_WAN        = 1 << 26 /* Support for native transport of 10GE_WAN_PHY on OC-192 */
};

```

3.3 Port Structure

Physical ports are reported as part of an array of struct ofp_phy_port in the OFPT_FEATURES_REPLY message. This spec extends the definition of phy ports – here we make a small concession: we use this struct to define switch internal ports and configured virtual ports as well as regular phy ports. In the interest of staying true to the packet switching spec, we retain the name ofp_phy_port.

```

/* Description of ports */
struct ofp_phy_port {
    uint16_t port_no;
    uint8_t hw_addr[OF_ETH_ALEN]; /* 00:00:00:00:00:00 if not an Ethernet port */
    uint8_t name[OF_MAX_PORT_NAME_LEN]; /* Null-terminated*/

    uint32_t config; /* Bitmap of OFPPC_* flags */
    uint32_t state; /* Bitmap of OFPPS_* flags */

    /* Bitmaps of OFPPF_* that describe features. All bits zeroed if
    * unsupported or unavailable. */
    uint32_t curr; /* Current features. */
    uint32_t advertised; /* Features being advertised by the port. */
    uint32_t supported; /* Features supported by the port. */
    uint32_t peer; /* Features advertised by peer. */

    uint16_t supp_swtype; /* Bitmap of switching type OFPST_* flags */
    uint16_t peer_swtype; /* Bitmap of peer's switching type */
    uint32_t supp_sw_tdm_gran; /* TDM switching granularity OFPTS_* flags */
    uint32_t peer_sw_tdm_gran; /* peer's TDM switching granularity */
    uint8_t pad[4]; /* Align to 64 bits */
    uint64_t bandwidth1; /* Bitmap of the OFPCBL_* or OFPCBT_* flags */
    uint64_t bandwidth2; /* Same type as bandwidth1 */
};
OFP_ASSERT(sizeof ( struct ofp_phy_cport) == 80);

```

The **port number** is a value that the datapath associates with a physical port. The port numbers usage has been changed to account for additional port types:

```
/* Port numbering. Physical ports are numbered starting from 0. */
enum ofp_port {
    /* Max number of real Ethernet and TDM ports – 0xfa00 (0x0000 to 0xf9ff)*/
    /* Switch internal ports - 0xfa00 to 0xfaff */
    /* Switch virtual circuit ports - 0xfb00 to 0xfeff*/
    /* Other virtual ports – 0xff00 to 0xffff*/

    OFPP_MAX                = 0xfa00,

    /* Fake output "ports". */
    OFPP_IN_PORT            = 0xffff, /* Send the packet out the input port. This
                                       virtual port must be explicitly used
                                       in order to send back out of the input port. */
    OFPP_TABLE              = 0xffff, /* Perform actions in flow table.
                                       NB: This can only be the destination
                                       port for packet-out messages. */
    OFPP_NORMAL             = 0xfffa, /* Process with normal L2/L3 switching. */
    OFPP_FLOOD              = 0xfffb, /* All physical ports except input port and
                                       Those disabled by STP. */
    OFPP_ALL                = 0xfffc, /* All physical ports except input port. */
    OFPP_CONTROLLER        = 0xfffd, /* Send to controller. */
    OFPP_LOCAL              = 0xfffe, /* Local openflow "port". */
    OFPP_NONE               = 0xffff /* Not associated with a physical port. */
};
```

The only change made from the OpenFlow packet switch specification is that we have limited the number of physical ports to 0xfa00 instead of 0xff00 and used the numbers in between to specify internal and virtual ports used in circuit switches. These internal port numbers can be used to specify “mapper” ports that map Ethernet packets to TDM time-slots, while the virtual ports can be used to define Virtual Concatenation Group (VCG) numbers used for VCAT technology in TDM switches.

The **hardware address** is the Ethernet address of the port for Ethernet ports, and is zeroed out for other types of ports (SONET/Wavelength) in circuit switches. The **name** field is a null terminated string containing a human readable name for the interface. In packet switches, examples are eth0, eth1 etc. In circuit switches, it can correspond to the standard Rack-Shelf-Slot-Port designation for telecom equipment. The **config** and **state** fields are currently the same as described in the OpenFlow specification for packet switches. The **features** bitmap has been modified to include line-rates in transport networks.

```
/* Features of physical ports available in a datapath. */
enum ofp_port_features {
    OFPPF_10MB_HD          = 1 << 0, /* 10 Mb half-duplex rate support. */
    OFPPF_10MB_FD          = 1 << 1, /* 10 Mb full-duplex rate support. */
    OFPPF_100MB_HD         = 1 << 2, /* 100 Mb half-duplex rate support. */
    OFPPF_100MB_FD         = 1 << 3, /* 100 Mb full-duplex rate support. */
    OFPPF_1GB_HD           = 1 << 4, /* 1 Gb half-duplex rate support. */
    OFPPF_1GB_FD           = 1 << 5, /* 1 Gb full-duplex rate support. */
};
```

```

OFPPF_10GB_FD      = 1 << 6, /* 10 Gb full-duplex rate support (10.3125 Gbps LAN PHY). */
OFPPF_COPPER       = 1 << 7, /* Copper medium */
OFPPF_FIBER        = 1 << 8, /* Fiber medium */
OFPPF_AUTONEG      = 1 << 9, /* Auto-negotiation */
OFPPF_PAUSE        = 1 << 10, /* Pause */
OFPPF_PAUSE_ASYM   = 1 << 11, /* Asymmetric pause */

```

```
/* The following have been added for WAN interfaces*/
```

```

OFPPF_X            = 1 << 20, /* Don't care – applicable to fiber switch ports */
OFPPF_OC1          = 1 << 21, /* 51.84 Mbps OC-1/STM-0 */
OFPPF_OC3          = 1 << 22, /* 155.52 Mbps OC-3/STM-1 */
OFPPF_OC12         = 1 << 23, /* 622.08 Mbps OC-12/STM-4 */
OFPPF_OC48         = 1 << 24, /* 2.48832 Gbps OC-48/STM-16 */
OFPPF_OC192        = 1 << 25, /* 9.95328 Gbps OC-192/STM-64 */
OFPPF_OC768        = 1 << 26, /* 39.81312 Gbps OC-768/STM-256 */
OFPPF_100GB        = 1 << 27, /* 100 Gbps */
OFPPF_10GB_WAN     = 1 << 28, /* 10 Gbps Ethernet WAN PHY (9.95328 Gbps) */
OFPPF_OTU1         = 1 << 29, /* OTN OTU-1 2.666 Gbps */
OFPPF_OTU2         = 1 << 30, /* OTN OTU-2 10.709 Gbps */
OFPPF_OTU3         = 1 << 31 /* OTN OUT-3 42.836 Gbps */

```

```
};
```

The above line rates are OCs (SONET standard) and their corresponding STMs (SDH standard). Optical Transport Network (OTN, G.709, digital wrapper) data rates have been added above as placeholders – this specification does not currently support OTN. The **swtype** fields are defined as:

```
/*Switching type of physical ports available in a datapath.*/
```

```

enum ofp_port_swtype {
    OFPST_L4      = 1 << 0, /* Capable of switching packets based on TCP or UDP headers*/
    OFPST_IP      = 1 << 1, /* Capable of switching packets based on IP headers */
    OFPST_MPLS    = 1 << 2, /* Capable of switching packets based on MPLS labels*/
    OFPST_VLAN    = 1 << 3, /* Capable of switching packets based on VLAN tags */
    OFPST_ETH     = 1 << 4, /* Capable of switching packets based on Ethernet headers */
    OFPST_T_SONET = 1 << 11, /* Capable of switching circuits (timeslots) based on SONET standard */
    OFPST_T_SDH   = 1 << 12, /* Capable of switching circuits (timeslots) based on SDH standard */
    OFPST_T_OTN   = 1 << 13, /* Capable of switching circuits (timeslots) based on OTN standard */
    OFPST_WAVE    = 1 << 14, /* Capable of switching circuits (wavelengths) based on ITU-T grid wavelengths */
    OFPST_FIBER   = 1 << 15 /* Capable of switching circuits (fibers) */

```

```
};
```

An OpenFlow packet switch can switch flows on the basis of Ethernet, IP, VLAN and transport layer headers. Such a switch will set multiple bits above. This specification does not currently support MPLS label switching or TDM switching based on OTN frame formats. The **sw_tdm_gran** fields are defined as:

```
/* Minimum switching granularity of TDM physical ports available in a datapath.*/
```

```

enum ofp_port_tdm_gran{
    OFPTSG_STS_1,      /* STS-1/STM-0 */
    OFPTSG_STS_3,      /* STS-3/STM-1 */
    OFPTSG_STS_3c,     /* STS-3c/STM-1*/
    OFPTSG_STS_12,     /* STS-12/STM-4 */
    OFPTSG_STS_12c,    /* STS-12c/STM-4c */
    OFPTSG_STS_48,     /* STS-48/STM-16 */
    OFPTSG_STS_48c,    /* STS-48c/STM-16c */
    OFPTSG_STS_192,    /* STS-192/STM-64 */
    OFPTSG_STS_192c,   /* STS-192c/STM-64c */

```

```

    OFPTSG_STG_768,      /* STS-768/STM-256 */
    OFPTSG_STG_768c    /* STS-768c/STM-256c */
}

```

The STS-*c signal bits should only be set if the switch supports contiguous concatenation in the switch capabilities. Note that the OpenFlow protocol does not support TDM signals smaller than STS-1 –i.e. no SONET VT’s or SDH LOVC’s are supported.

The **bandwidth1** and **bandwidth2** fields of the ofp_phy_cport struct need to be explained in more detail. Their purpose is to *flexibly indicate the bandwidth supported and currently used/available* by the circuit port. Their interpretation depends on the switching type of the switch port.

- If the switching type of the port (defined in the supp_swtype field) is OFPST_WAVE, then bandwidth1 and bandwidth2 are bitmaps corresponding to the OFPCBL_* flags. Additionally, bandwidth1 identifies the wavelengths supported by the switch and bandwidth2 identifies the wavelengths currently under use (ie. cross-connected)
- If the switching type of the port is OFPST_T_SONET or OFPST_T_SDH, then bandwidth1 and bandwidth2 identify available time slots for various TDM signals. Depending on the line-rate, we define different interpretations of the bits in the 64-bit field below. Note that we do not need one of the fields to identify supported time-slots as that can be inferred from the line-rate and TDM switching granularity.

For a switching type of OFPST_WAVE, bandwidth 1 has the following meaning: The lower 10 bits of the 64 bit uint64_t will be used for flags with special meaning. The upper 54 bits will be used to designate ITU-T grid frequencies supported by the switch port.

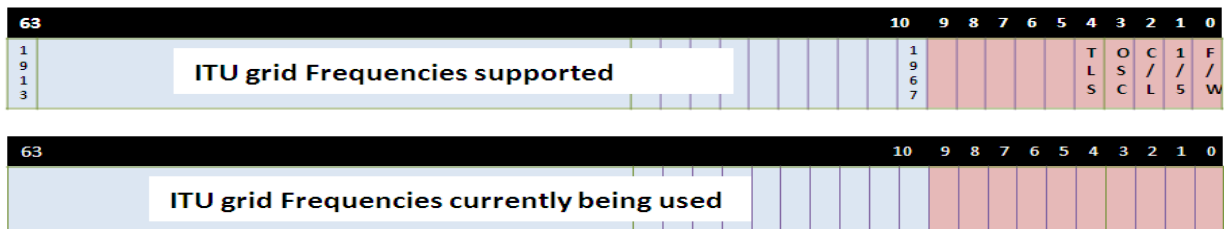
```

/*Switching granularity of TDM physical ports available in a datapath. */
enum ofp_port_lam_bw{
    OFPCBL_X          = 1 << 0, /* 1 if fiber switch – don’t care what wavelength is used, 0 if lambda switch */
    OFPCBL_100_50     = 1 << 1, /* 1 if 100GHz channel spacing, 0 if 50 GHz */
    OFPCBL_C_L        = 1 << 2, /* 1 if C –band, 0 if L-band */
    OFPCBL_OSC        = 1 << 3, /* 1 if supporting the OSC at 1510nm, 0 if not */
    OFPCBL_TLS        = 1 << 4 /* 1 if using TLS, 0 if not */
};

```

Bits 5 through 9 are reserved for future use. In a 100 GHz channel spaced system, the bits 10-63 are as follows: bit 10 corresponds to 196.7 THz (1524.11 nm), bit 11 to 196.6 THz, bit 12 to 196.5 THz and so on, with bit 63 corresponding to 191.4 THz (1566.33 nm). In an L-Band system, bit 10 would correspond to 190.7 THz (1572.06 nm) and bit 63 to 185.4 THz (1617.08 nm).

bandwidth1 and bandwidth2 fields:



3.4 Circuit Flow Cross-Connect Structure

The logical equivalent of the `ofp_match` structure from the packet switching spec is the `ofp_connect` structure in the circuit switching addendum. It is used to describe the circuit flow much like the match structure is used to describe the packet flow. When describing a cross-connection, the following `ofp_connect` is used within `struct ofp_cflow_mod` message (Section 3.5):

```
/*Description of a cross-connection*/
struct ofp_connect{
    uint16_t wildcards;           /* identifies which two ports to use below */
    uint16_t num_components;     /* identifies number of cross-connect to be made – ie. num array elems*/
    uint8_t pad[4];             /* for 64 bit alignment*/

    uint16_t in_port[0];        /* OFPP_* ports – real or virtual */
    uint16_t out_port[0];       /* OFPP_* ports – real or virtual */

    struct ofp_tdm_port in_tport[0]; /* description of TDM channel */
    struct ofp_tdm_port out_tport[0];

    struct ofp_wave_port in_wport[0]; /* description of lambda channel */
    struct ofp_wave_port out_wport[0];
};
OFP_ASSERT(sizeof(struct ofp_connect) == 8);
```

The **wildcards** field simply identifies which fields in the `ofp_connect` structure should be ignored when looking to cross-connect an incoming port to an out-going port. Note that of the 6 choices currently defined in the `wildcards` field, *at least 4 need to be set*, so that the 2 zeroed out wildcard flags correspond to a type of input port and a type of output port. Since the `ofp_connect` structure is really like a header followed by variable length arrays, the **num_components** field identifies the bytes to follow.

```
/* Flow wildcards */
enum ofp_connect_wildcards {
    OFPCW_IN_PORT      = 1 << 0,
    OFPCW_OUT_PORT     = 1 << 1,
    OFPCW_IN_TPORT    = 1 << 2,
    OFPCW_OUT_TPORT   = 1 << 3,
    OFPCW_IN_WPORT    = 1 << 4,
    OFPCW_OUT_WPORT   = 1 << 5
};
```

For example, a valid wildcard entry can be `0x003C` indicating a regular input port to output port connection. The `num_components` identify in this case could be 2, indicating that the 4 bytes immediately following the padding bytes, contain ports numbers for 2 **in_ports**, followed by 4 bytes for two **out_ports**. A TDM to TDM cross-connection can be identified as `0x0033` and a lambda cross-connection as `0x000F`. In the former case, if `num_components` is 2, it indicates that the 16 bytes following the padding bytes, describe two **in_tports**, followed by 16 bytes of two **out_tports**. It is also possible to connect dissimilar port types as will be explained in the next section. As an example a wildcard entry of `0x0036` would indicate cross-connecting a regular input port (real or virtual) to a TDM port. Descriptions of TDM and wavelength ports are defined below;

```

/* Description of a TDM port */
struct ofp_tdm_port {
    uint16_t tport;           /* port numbers in OFPP_* ports */
    uint16_t tstart;         /* starting time-slot in binary */
    uint32_t tsignal;        /* one of OFPTSG_* flags */
};
OFP_ASSERT(sizeof(struct ofp_tdm_port) == 8);

/*Description of a wavelength port */
struct ofp_wave_port {
    uint16_t wport;          /* restricted to real port numbers in OFPP_* ports */
    uint8_t pad[6];
    uint64_t wavelength;    /* use of the OFPCBL_* flags */
};
OFP_ASSERT(sizeof(struct ofp_wave_port) == 16);

```

As mentioned above, multiple ports can be cross-connected on the switch with a single struct ofp_connect. For example if we wish to make the following connections:

```

tport = 1, tsignal = STS-3c, tstart = 9  ↔  tport = 3, tsignal = STS-3c, tstart = 9
tport =5, tsignal = STS-12c, tstart = 24 ↔  tport = 3, tsignal = STS-12c, tstart = 24
tport = 1, tsignal = STS-3c, tstart = 12  ↔  tport = 3, tsignal = STS-3c, tstart = 0

```

We can do so by defining the num_components as 3, creating an array of ofp_tdm_port's with the left side of the above connections as in_port, and the right side as out_port and requiring that corresponding elements of the two arrays should be cross-connected.

3.5 Circuit Flow Add/Modify/Delete

Modifications to the flow table in a circuit switch are accomplished via the OFPT_FLOW_MOD message. This message is the only new message type introduced by this spec. The enum ofp_type is changed as below.

```

enum ofp_type {
    /* Immutable messages. */
    OFPT_HELLO,           /* Symmetric message */
    OFPT_ERROR,           /* Symmetric message */
    OFPT_ECHO_REQUEST,    /* Symmetric message */
    OFPT_ECHO_REPLY,      /* Symmetric message */
    OFPT_VENDOR,          /* Symmetric message */

    /* Switch configuration messages. */
    OFPT_FEATURES_REQUEST, /* Controller/switch message */
    OFPT_FEATURES_REPLY,   /* Controller/switch message */
    OFPT_GET_CONFIG_REQUEST, /* Controller/switch message */
    OFPT_GET_CONFIG_REPLY, /* Controller/switch message */
    OFPT_SET_CONFIG,       /* Controller/switch message */

    /* Asynchronous messages. */
    OFPT_PACKET_IN,        /* Async message */
    OFPT_FLOW_EXPIRED,     /* Async message */
    OFPT_PORT_STATUS,      /* Async message */

    /* Controller command messages. */

```

```

OFPT_PACKET_OUT,          /* Controller/switch message */
OFPT_FLOW_MOD,           /* Controller/switch message */
OFPT_PORT_MOD,           /* Controller/switch message */

/* Statistics messages. */
OFPT_STATS_REQUEST,      /* Controller/switch message */
OFPT_STATS_REPLY,       /* Controller/switch message */

/* Controller command messages for circuit switches. */
OFPT_CFLOW_MOD           /* Controller/switch message */
};

```

The associated struct `ofp_cflow_mod` is shown below:

```

/* Circuit flow setup, modification and teardown (controller → datapath) */
struct ofp_cflow_mod {
    struct ofp_header header;
    uint16_t command;          /* one of OFPPC_* flags */
    uint16_t hard_timeout;     /* max time to connection tear-down,
                               /* if 0 then explicit tear-down required */

    uint8_t pad[4];
    struct ofp_connect connect; /* 8B followed by variable length arrays */
    struct ofp_action_header actions[0]; /* variable number of actions*/
};
OFP_ASSERT(sizeof(struct ofp_cflow_mod) == 24);

```

The length field in the **header** is the total length of the message. The **connect** field describes the cross-connections to be made and is of variable length. The number of bytes defining the cross-connections to be made can be inferred from the wildcards and `num_components` field in struct `ofp_connect` as described in the previous section. The **actions** length can then be inferred by subtracting all preceding bytes from the length field in the header. The **command** field must be one of the following as defined in OF packet switch spec v0.8.9:

```

enum ofp_flow_mod_command {
    OFPFC_ADD,          /* New flow. */
    OFPFC_MODIFY,       /* Modify all matching flows. */
    OFPFC_MODIFY_STRICT, /* Modify entry strictly matching wildcards */
    OFPFC_DELETE,       /* Delete all matching flows, analogous to tearing down a circuit flow. */
    OFPFC_DELETE_STRICT, /* Strictly match wildcards and priority. */
    OFPFC_DROP          /* Terminate a circuit flow */
};

```

However, for circuit flows, we *require* that `OFPFC_MODIFY_STRICT` and `OFPFC_DELETE_STRICT` are used to modify and terminate existing connections. Also note that there is no ‘idle timeout’ field in the `flow_mod` struct as it normally not possible to tell in circuit switches, if the circuit is idle or not. We do however include a **hard_timeout** field, which may find some uses in certain cases, where the duration of a flow is pre-determined. If this value is set to zero, circuit flows will be permanent and an explicit `DELETE_STRICT` will be required to teardown the flow. Additionally, we have added an extra command `OFPFC_DROP`, which is used to signify a termination of a circuit flow and subsequent extraction of packet flows from it. This is accompanied by the `ofp_action_ckt_input` struct described in the next section. Again, the use of the ‘drop’ terminology is common in transport networks to mean termination of a circuit trail, not the loss of the circuit or packet flow.

3.6 Circuit Flow Action Types

The currently defined action types are:

```
enum ofp_action_type {
    OFPAT_OUTPUT,          /* Output to switch port. */
    OFPAT_SET_VLAN_VID,   /* Set the 802.1q VLAN id. */
    OFPAT_SET_VLAN_PCP,   /* Set the 802.1q priority. */
    OFPAT_STRIP_VLAN,     /* Strip the 802.1q header. */
    OFPAT_SET_DL_SRC,     /* Ethernet source address. */
    OFPAT_SET_DL_DST,     /* Ethernet destination address. */
    OFPAT_SET_NW_SRC,     /* IP source address. */
    OFPAT_SET_NW_DST,     /* IP destination address. */
    OFPAT_SET_TP_SRC,     /* TCP/UDP source port. */
    OFPAT_SET_TP_DST,     /* TCP/UDP destination port. */
    OFPAT_CKT_OUTPUT,     /* Output to circuit port */
    OFPAT_CKT_INPUT,      /* Input from circuit port*/
    OFPAT_VENDOR = 0xffff
};
```

Currently the use of actions in the OpenFlow protocol extensions for circuit switching is limited – we expect this to change as the protocol develops. This specification adds actions related to circuit ports: specifically `OFPAT_CKT_INPUT` and `OFPAT_CKT_OUTPUT`. The usage of these actions together with the corresponding flow mod messages they are part of is explained in the context of inserting and extracting packet flows from circuit flows.

3.6.1 Adapting packet flows to circuit flows:

Packet switches that operate in the WAN often have linecard ports with circuit features. For example, backbone or access routers have SONET/SDH ports where packets are adapted and inserted into SONET TDM frames for transport over a SONET ring or point-to-point network. Additionally, modern circuit switches have packet interfaces on which they can accept incoming packets (Ethernet frames) and adapt them to forward out of the circuit interfaces or other packet interfaces, the latter ability enabled with a packet switching fabric in addition to the traditional circuit switching fabric. For both kinds of switches, a hardware packet flow table exists, as defined in the OpenFlow Switch Specification for packet switches. Incoming packet flows are matched according to the 10-tuple packet headers and corresponding actions are performed to the matching flows. To adapt packet flows to circuit flows we define a new action struct in line with the other flow action structures defined in sec 5.2.3 of the OpenFlow Switch Specification v0.8.9. For convenience we repeat the `ofp_action_header`

```
/* Action header that is common to all actions. The length includes the
 * header and any padding used to make the action 64-bit aligned.
 * NB: The length of an action *must* always be a multiple of eight. */
struct ofp_action_header {
    uint16_t type;          /* One of OFPAT_*. */
    uint16_t len;          /* Length of action, including this
                           * header. This is the length of action,
                           * including any padding to make it 64-bit aligned. */
    uint8_t pad[4];
};
OFP_ASSERT(sizeof(struct ofp_action_header) == 8);
```

For the action type `OFPAT_CKT_OUTPUT`, the corresponding struct has the following fields

```
/* Action structure for OFPAT_CKT_OUTPUT, which sends packets out of a ckt_port */
struct ofp_action_ckt_output {
    uint16_t type;           /* OFPAT_CKT_OUTPUT */
    uint16_t len;           /* Length is 24*/
    uint16_t adaptation;    /* Adaptation type – one of OFPCAT */
    uint16_t cport;        /* Real or virtual OFPP_* ports */

    /* Define the circuit port characteristics if necessary*/
    uint64_t lambda;       /* use of the OFPCBL_* flags */
    uint32_t tsignal;      /* one of OFPTSG_* flags. Not valid if ofp_connect defines TDM signal */
    uint16_t tstart;       /* starting time-slot in binary. Not valid if ofp_connect defines TDM signal */
    uint16_t tlcas_enable  /* enable/disable Link Capacity Adjustment Scheme (LCAS) */
};
OFP_ASSERT(sizeof(struct ofp_action_ckt_output) == 24);
```

If the output circuit port is defined (in `opf_phy_port`) with a switching type of `OFST_WAVE`, and if the **adaptation** is not to a TDM frame type, then it should ignore the TDM related fields above. The current adaptation types are defined below:

```
enum ofpc_adap_type {
    OFPCAT_NONE           = 1 << 0,    /* no adaptation – eg. native transport of GE or 10GE LAN-PHY*/
    OFPCAT_POS            = 1 << 1,    /* Packet-over-SONET/SDH adaptation */
    OFPCAT_GFP            = 1 << 2,    /* Generic Framing Procedure adaptation to SONET/SDH */
    OFPCAT_10G_WAN        = 1 << 3,    /* 10G Ethernet WAN PHY framing for SONET OC-192
                                        * or SDH STM-64*/
};
```

As a special case, consider a TDM switch with 1) Ethernet interfaces and the ability to adapt Ethernet packets to SONET frames and 2) the ability to perform a SONET technology known as Virtual Concatenation (VCAT) whereby multiple TDM signals are virtually concatenated and regarded as a single signal with the cumulative bandwidth of its component signals. The OpenFlow controller can, via an `OFP_FLOW_MOD` message, define flows on the packet flow table with `OFPAT_CKT_OUTPUT` action for matching packet flows, where the **cport** field is a virtual port (in the range `0xfb00` to `0xfeff`). This virtual port can correspond to the Virtual Concatenation Group (VCG) number.

Then in the circuit switching flow table (cross-connect table) it can define the VCG via an `OFP_CFLOW_MOD` message. VCGs are typically defined using internal switch ports. The OpenFlow controller has full visibility into these internal ports as the controller needs to have the ability to add/remove VCGs as well as modify them via adding/removing the component members of the VCG.

3.6.2 VCG operations:

There are five operations of interest in the use of VCGs – creating a VCG, deleting a VCG, adding members (component TDM signals) to the VCG, removing members from the VCG, and modifying VCG parameters. Additionally if LCAS is supported on the switch, then adding or removing member component signals to the VCG can be hitless to the traffic being supported by the VCG. All VCG operations are performed with the `OFP_CFLOW_MOD` message. Details are given below:

- Creating a VCG: done with OFP_CFLOW_MOD message, where the inbuilt ofp_connect struct defines the VCG component signals. The connect struct's num_components identify the number of component signals in the VCG. For example a wildcard entry of 0x0036 can be used to identify the cross-connection of in_port (the VCG) and out_tport (the internal port, signal, and starting time-slot to which the VCG is mapped).

```

in_port [0] = 0xfd11  ↔ out_tport [0] = tport = 3, tsignal = STS-3c, tstart = 9
           [1]= 0xfd11  ↔           [1]= tport = 4, tsignal = STS-12c, tstart = 24
           [2]= 0xfd11  ↔           [2]= tport = 1, tsignal = STS-3c, tstart = 0

```

Similarly a wildcard field of 0x0039 can be used to identify the cross-connection of out_port (the VCG) and in_tport (the internal port). The command is OFPFC_ADD, hard_timeout is controller's choice, and an action struct MUST exist of type OFPAT_CKT_OUTPUT or OFPAT_CKT_INPUT depending on ingress/egress switch (fields are the same). Within the action struct, tlcas_enable and adaptation are specified and the cport is internal port number to which the VCG is mapped. A VCG with no members can also be created in which case a) wildcards should be 0x003E or 0x003D b) num_components should be 1 and c) in_port[0] should identify the VCG virtual port number and d) cport within the action struct is ignored. If the VCG cannot be created for any reason, an error message is generated. An additional OFP_CFLOW_MOD message needs to be sent to cross-connect the VCG signals (internal port time-slots) to physical TDM ports and signals.

- Adding members to VCG – done with OFP_CFLOW_MOD message, where the command used is OFPFC_MODIFY_STRICT. With the wildcard as 0x0036 or 0x0039, the VCG is identified with the VCG port number as in_port or out_port in struct ofp_connect. The array out_tport (or in_tport) specifies ONLY the members to add – existing members should not be repeated here. However if the controller tries to add an existing member no errors are generated.
 - An action struct is not necessary when using the modify message for this purpose. If an action struct exists, it MUST specify absolute values of tlcas_enable, adaptation and cport and these will be applied to the whole VCG.
 - If the VCG does not exist, a new one will be created in which case the action struct should be there there - if not an error message is generated. No default values are ever assumed
- Modifying action parameters - done with command OFPFC_MODIFY_STRICT. VCG is identified with VCG number as in_port or ou_port in connect struct. An action struct must exist and it MUST specify absolute values of tlcas_enable, adaptation and cport and these will be applied to the whole VCG.
 - If the VCG does not exist, a new one will be created with 0 or more members depending on choice of wildcards used.
 - If a OFPFC_MODIFY_STRICT is sent with no action struct and wildcards such that no members are to be added, then the VCG cannot be created and an error messages is generated
- Deleting a VCG - done with command OFPFC_DELETE_STRICT and all that is needed is the VCG number in the ofp_connect struct. The wildcard field should be 0x003E or 0x003D depending on ingress or egress VCG. No actions are needed. If actions exist they should be ignored. If the VCG does not exist, no error message is generated

- Deleting members from VCG – done with command OFPFC_DELETE_STRICT. VCG is identified with VCG number as in_port or out_port in ofp_connect. The wildcard field should not be 0x003E or 0x003D otherwise the entire VCG will be deleted. The array out_tport (or in_tport) specifies ONLY the members to delete – should be an existing member but if not, then no error message is generated. An action struct should not be included in the delete message – if actions exist they should be ignored. Deleting the last member of the VCG should NOT delete the VCG. If the VCG does not exist no error messages are generated.

In general, when adding or modifying members, the time-slots specified MUST be free – if not an error is generated. In other words, timeslots must be released (deleted) before being re-assigned. When making regular TDM to TDM time-slot crossconnects, only ADD and DELETE_STRICT apply.

3.6.3 Extracting Packet Flows from Circuit Flows:

To go back from circuit to packet flows, we need to terminate the circuit flow, de-encapsulate the payload if an adaption was used and send out the packets to packet interfaces or do flow matching if a packet switching fabric exists in the same switch. Often in a circuit switch the termination of a circuit flow is also accomplished by defining a cross-connection to a ‘Drop’ port. This can be accomplished with an OFP_CFLOW_MOD message with associated command OFPFC_DROP. Note that this use of ‘drop’ does not have the same meaning as ‘dropping a packet’. A cross-connection to a Drop port can also be defined with a struct ofp_connect with an associated action struct as defined below:

```
struct ofp_action_ckt_input {
    uint16_t type;           /* OFPAT_CKT_INPUT
    uint16_t len;           /* Length is 24*/
    uint16_t adaptation;    /* Adaptation type – one of OFPCAT */
    uint16_t cport;        /* Real or virtual OFPP_* ports */

    /* Define the circuit port characteristics if necessary*/
    uint64_t lambda;       /* use of the OFPCBL_* flags */
    uint32_t tsignal;      /* one of OFPTSG_* flags. Not valid if ofp_connect defines TDM signal */
    uint16_t tstart;       /* starting time-slot in binary. Not valid if ofp_connect defines TDM signal */
    uint16_t tlcas_enable  /* enable/disable Link Capacity Adjustment Scheme (LCAS) */
};
OFP_ASSERT(sizeof(struct ofp_action_ckt_input) == 24);
```

If the switch has a packet switching fabric, then the packets extracted from the circuit flow can be matched to packet flow table entries. In this case the input port for the packets can be the virtual port defined in the ofp_action_ckt_input struct.

3.7 Error Messages

A new error message is defined with associated error codes for reporting problems with circuit flow setup.

```
enum ofp_error_type {
    OFPET_HELLO_FAILED,      /* Hello protocol failed. */
    OFPET_BAD_REQUEST,      /* Request was not understood. */
};
```

```

    OFPET_BAD_ACTION,          /* Error in action description. */
    OFPET_FLOW_MOD_FAILED,    /* Problem modifying flow entry. */
    OFPET_CFLOW_MOD_FAILED    /* Problem modifying circuit flow entry. */
};

/* ofp_error_msg 'code' values for OFPET_CFLOW_MOD_FAILED. 'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_cflow_mod_failed_code {
    OFPCFMFC_VCG_DEF, /* creation or modification of VCG failed */
    OFPCFMFC_OVERLAP, /* Attempted to add overlapping flow with currently used time-slot*/
    OFPCFMFC_MISMATCH /* Mismatched tsignals in ofp_connect struct */
};

```

The data field contains at least 64 bytes of the failed request. The most relevant error code should be sent. For example if the VCG creation failed due to an attempt to use an already allocated mapper time slot, OVERLAP code should be specified.