

Extensions to OpenFlow to support Circuit Switching

Addendum to OpenFlow Specification (v0.8.9) – vc0.1

May 7, 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 latest version of the OpenFlow switch specification for packet switches (v0.8.9) 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 or fibers. It also covers the OpenFlow protocol changes required to manage an OpenFlow circuit switch from a remote controller. This document should be viewed as an addendum to the switch specification for packet switches and not independently – this is because this document also applies to packet switches with circuit ports (like backbone routers) as well as circuit switches with packet switching capability on some ports.

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 CS 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

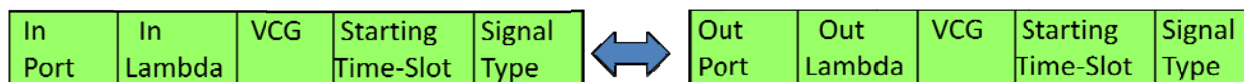


Fig. 1: Circuit flow table (cross-connect table) entry

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 OFCs protocol. 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. 2.

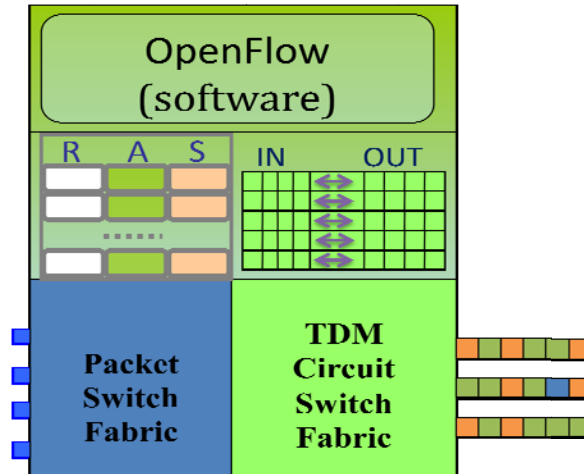


Fig 2: An OpenFlow switch with packet and circuit hardware flow tables

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. We detail these structures below

3.1 Circuit Port Structure

Physical circuit ports are defined with the following structure:

```

/* Description of a physical circuit port */
struct ofp_phy_cport {
    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;              /* Bitmap of TDM switching granularity OFPTSG_* flags */
    uint32_t peer_sw_tdm_gran;              /* Bitmap of peer's switching granularity */
    uint64_t bandwidth1;                    /* Bitmap of the OFPCBL_* or OFPCBT_* flags */
    uint64_t bandwidth2;                    /* Same type as supp_bandwidth1 */
};
OFP_ASSERT(sizeof ( struct ofp_phy_cport) == 92);

```

The ofp_phy_cport structure is similar to the ofp_phy_port struct for packet switches (sec 5.2.1 of the OF packet switch spec v0.8.9). The port number uses the same flags in both structures. The **port number** is a value that the datapath associates with a physical port. The port numbers use the following convention:

```

/* Port numbering. Physical ports are numbered starting from 0. */
enum ofp_port {
    /* Maximum number of physical switch ports. */
    /* 0xfd01 to 0xff00 reserved for virtual circuit ports like VCG
    OFPP_MAX          = 0xfd00,

    /* 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       = 0xffff9, /* Perform actions in flow table.
                                NB: This can only be the destination
                                port for packet-out messages. */
    OFPP_NORMAL      = 0xffffa, /* Process with normal L2/L3 switching. */
    OFPP_FLOOD       = 0xffffb, /* All physical ports except input port and
                                Those disabled by STP. */
    OFPP_ALL         = 0xffffc, /* All physical ports except input port. */
    OFPP_CONTROLLER  = 0xffffd, /* Send to controller. */
    OFPP_LOCAL       = 0xffffe, /* Local openflow "port". */
    OFPP_NONE        = 0xfffff /* 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 0xfd00 instead of 0xff00 and used the 512 port numbers in between to specify virtual ports used in circuit switches. These virtual ports can be the Virtual Concatenation Group (VCG) numbers used for VCAT technology in TDM switches. They can also be used to specify virtual ports in transitioning from physical circuit ports to physical packet ports. Both uses will be explained in later sections.

The **hardware address** is the Ethernet address of the port for Ethernet ports, and can be ignored for other types of ports (SONET/Wavelength) in circuit switches. When ignoring such a port the field can be zeroed out. 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 Shelf: Rack: 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 data 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, /* TCP or UDP port based packet switch */
    OFPST_IP      = 1 << 1, /* IP packet switch */
    OFPST_MPLS    = 1 << 2, /* MPLS labels switch*/
    OFPST_VLAN    = 1 << 3, /* VLAN packet switch */
    OFPST_ETH     = 1 << 4, /* Ethernet packet switch */
    OFPST_T_SONET = 1 << 5, /* TDM switch SONET standard */
    OFPST_T_SDH   = 1 << 6, /* TDM switch SDH standard */
    OFPST_T_OTN   = 1 << 7, /* TDM switch OTN standard */
    OFPST_WAVE    = 1 << 8, /* Wavelength switch */
    OFPST_FIBER   = 1 << 9 /* Fiber switch */

```

```

};

```

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:

```

/*Switching granularity of TDM physical ports available in a datapath. */

```

```

enum ofp_port_tdm_gran{
    OFPTSG_STS-1  = 1 << 0, /* STS-1/STM-0 */
    OFPTSG_STS-3  = 1 << 1, /* STS-3/STM-1 */
    OFPTSG_STS-3c = 1 << 2, /* STS-3c/STM-1 */
    OFPTSG_STS-12 = 1 << 3, /* STS-12/STM-4 */
    OFPTSG_STS-12c = 1 << 4, /* STS-12c/STM-4c */
    OFPTSG_STS-48 = 1 << 5, /* STS-48/STM-16 */
    OFPTSG_STS-48c = 1 << 6, /* STS-48c/STM-16c */
    OFPTSG_STS-192 = 1 << 7, /* STS-192/STM-64 */
    OFPTSG_STS-192c = 1 << 8, /* STS-192c/STM-64c */

```

```

    OFPTSG_STIS-768          = 1 << 9, /* STS-768/STM-256 */
    OFPTSG_STIS-768c       = 1 << 10 /* 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 191.3 THz (1567.13 nm). In an L-Band system, bit 10 would correspond to 109.7 THz (1572.06 nm) and bit 63 to 185.3 THz (1617.88 nm). Although bit 1 is used to identify a 100GHz system or a 50GHz system, this specification currently does not support a 50GHz spaced system. Bit 2 identifies a C or L band system. A wavelength switch has mux/demux filters that are designed to operate in either the C or L bands but not both. Accordingly this bit indicates whether the flags in bit 10-63 correspond to C or L band wavelengths.

Bit 0 is used to identify a fiber switch as opposed to a wavelength switch which can distinguish between and switch wavelengths individually. A fiber switch is typically agnostic to whatever signals are carried in the incoming fiber and thus a ‘don’t care’ applies in this case. However if a fiber switch is used with a

transponder on the port, then the switch port should be treated as a wavelength switch port (bit 0 = 0, one of bits 10-63 should be set, and the data rate flag OFPPF_* should be set).

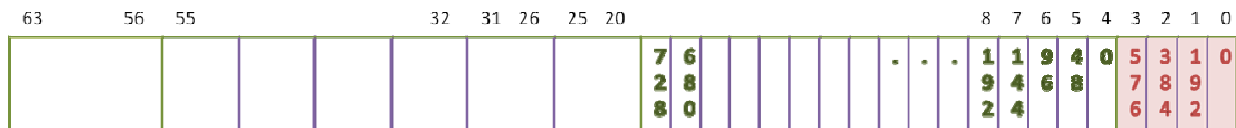
Another relevant situation is with regards to a wavelength switch used in a system where different wavelengths support different line-rates. For example, in a 40 channel C-band system, it is possible (though unlikely) that some wavelengths have transceivers running at 2.5 Gbps and others at 10 Gbps. While it may seem that a wavelength switch should be agnostic to the line-rate akin to fiber switches without transponders, this is not always true. This is because wavelength switches have filters to mux/demux wavelengths and these filters have passband widths designed for a certain line-rate and channel spacing. While a filter designed for 10 Gbps signal can pass a 2.5 Gbps signal (albeit with more noise) it cannot pass a 40 Gbps signal. Thus for a wavelength switch, we need to specify for the port the highest line-rate it can support with the OFPPF_* flags.

Bit 3 is used to identify the Optical Supervisory Channel (OSC-typically at 1510 nm) which is converted to the electrical domain, processed and then re-converted to the optical domain for transmission (a process known as OEO). Bit 4 is used to identify a Path Terminating Equipment (PTE) which has a tunable laser source (TLS) on the output port. In this case, the bandwidth1 field identifies the tuning range of the TLS and the bandwidth2 field identifies the current laser wavelength. Note that the TLS must support ITU grid wavelengths. Also our use of 'PTE' here signifies equipment that adapts packets to circuits and vice versa – for eg. a large backbone packet switch with circuit ports.

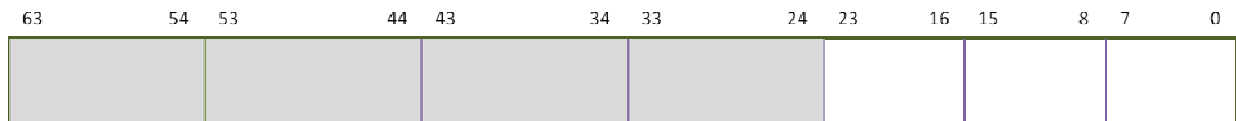
For a switching type of OFPST_T_SONET_or_OFPST_T_SDH, bandwidth1 and bandwidth2 fields should be used together to identify the available *starting* time slots on the optical carrier. This then depends on the line-rate of the optical carrier and the minimum switching granularity of the switch. Assuming STS-1 for the latter, we identify the bit interpretation in the two fields as below:

OC-768

Bandwidth1 field:



Bandwidth2 field:

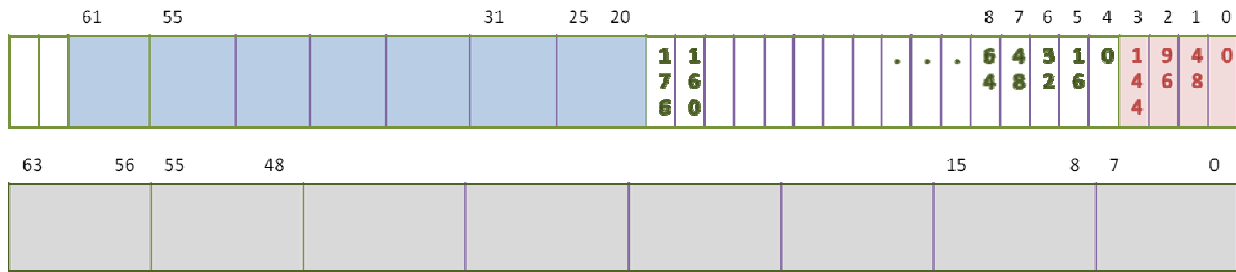


For an OC-768 line-rate, there is 1 possible *starting timeslot* for an STS-768 signal, 4 possible slots for an STS-192c signal, 16 possible slots for an STS-48c signal, 64 for STS-12c, 256 for STS-3c and 768 for STS-1. Of these, the bandwidth1 field identifies all possible choices for STS-768, STS-192c and STS-48c signals.

The availability of time-slots for the STS-768 signal is shown by setting all four bits 0-3. For STS-192c and STS-48c the availability of the time-slot is indicated by setting the corresponding bit or zeroing it if it is already being used. Bits 20-55 indicate 6 possible choices of available starting time-slots for an STS-12 signal (out of a total 64). The interpretation of these bits merits a more detailed explanation. When we say 64 possible starting time-slot choices, we mean that choice 0 is time-slot 0, choice 1 is time-slot 12, and choice 2 is time-slot 24 and so on. The 64 choices can be represented in binary with 6 bits. Thus if bits 31-26 of the bandwidth1 field are for example 110001, it represents the 49th choice of starting time slot – ie. time-slot 588. Similarly, bits 56-63 of bandwidth1 and 0-23 of bandwidth2 represent 4 possible starting time-slot choices (8 bits each) for an STS-3c signal (out of a possible 256), and bits 24-63 represent 4 possible choices (10 bits each) for an STS-1 signal.

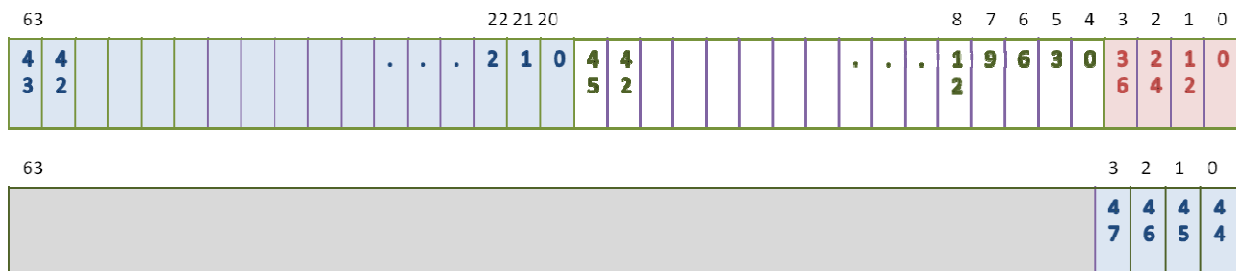
Lastly it should be noted, that while the number of choices for the lower rate signals appears low, this is generally not the case. This is because any choice for a higher rate signal represents multiple choices for a lower rate signal. For example, if as above the 49th choice is identified for an STS-12c signal (starting time-slot 588), it also represents 4 additional choices for an STS-3c signal – 588, 591, 594, and 597, as well as 12 additional choices for an STS-1 signal – 588, 589, 590 and so on. Similar reasoning can be followed for OC-192 below:

OC-192



Bits 20-61 in bandwidth1 represent 7 choices for an STS-3c signal (out of 64), and bandwidth2 represents 8 choices for an STS-1 signal (out of 192).

OC-48



With an OC-48 signal, we can represent all 48 possible time-slots for STS-1 signals, as well as all 16 STS-3cs, and 4 STS-12cs. Similarly we can represent all possible choices for all possible signals in an OC-12 signal with just the bandwidth1 field.

3.2 Circuit Flow Cross-Connect Structure

When describing a cross-connection, the following struct `ofp_connect` is used within struct `ofp_cflow_mod` message (Section 3.4):

```
/*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*/

    uint16_t  in_port;            /* OFPP_* ports – real or virtual */
    uint16_t  out_port;           /* 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, exactly 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.

```
/* 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
};
```

Examples of valid wildcard entries can be 0x003C indicating a regular input port to output port connection. A TDM to TDM cross-connection can be identified as 0x0033 and a lambda cross-connection as 0x000F. It is also possible to connect dissimilar port types as will be explained in the next section. As an example a wildcard entry of 0x0006 would indicate cross-connecting a regular input port (real or virtual) to a TDM output port. Descriptions of TDM and wavelength ports are defined below;

```
/* Description of a TDM port */
struct ofp_tdm_port {
    uint16_t  tport;           /* restricted to real port numbers in OFPP_* ports */
    uint32_t  tsignal;        /* one of OFPTSG_* flags */
    uint16_t  tstart;         /* starting time-slot in binary */
};
```

```
/*Description of a wavelength port */
struct ofp_wave_port {
    uint16_t  wport;          /* restricted to real port numbers in OFPP_* ports */
    uint64_t  wavelength     /* use of the OFPCBL_* flags */
};
```

Note that 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:

```
tpport = 1, tsignal = STS-3c, tstart = 9  ↔  tport = 3, tsignal = STS-3c, tstart = 9
tpport =5, tsignal = STS-12c, tstart = 24 ↔  tport = 3, tsignal = STS-12c, tstart = 24
tpport = 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.

3.3 Packet Flow and Circuit Flow Interface

3.3.1 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, some 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);
```

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
};

```

This specification adds actions related to circuit ports. 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 */
* NB: This action should be used within an ofp_flow_mod struct for packet switching fabrics and not the *
* ofp_cflow_mod struct for circuit switching fabrics
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 */
    uint16_t tstart;       /* starting time-slot in binary */
    uint8_t pad[2];
};
OFP_ASSERT(sizeof(struct ofp_action_ckt_output) == 24);

```

If the output circuit port is defined with a switching type of OFPST_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, a TDM switch with the ability to perform Virtual Concatenation (VCAT) can on the packet flow table define a struct ofp_action_ckt_output for matching packet flows, where the cport field is a virtual port (between 0xfd01 to 0xff00). 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 cross-connect using struct ofp_connect with an in_port field with the same virtual port number, and an out_tport array of struct ofp_tdm_port, which would have the individual component signals of the VCG. As an example, with a wildcard entry of 0x0006 in struct ofp_connect and a num_component = 3, we can define the following VCG

```

in_port = 0xfd11  ↔ tport = 3, tsignal = STS-3c, tstart = 9
                  → tport = 4, tsignal = STS-12c, tstart = 24
                  → tport = 1, tsignal = STS-3c, tstart = 0

```

Here in_port is a virtual port (between 0xfd01 and 0xff00) corresponding to the VCG number, and the component signals are defined as array elements of the out_tport array.

3.3.2 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. 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 (sec 3.2) with an associated action struct as defined below:

```
struct ofp_action_ckt_input {
    uint16_t type;           /* OFPAT_CKT_INPUT
    uint16_t len;           /* Length is 8*/
    uint16_t adaptation;    /* Adaptation type – one of OFPCAT */
    uint16_t cport;        /* Real or virtual OFPP_* ports */
};
```

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.

3.4 Add/Modify/Delete Circuit Flows

Modifications to the flow table in a circuit switch are accomplished via the OFPT_FLOW_MOD message. The associated struct is shown below:

```
/* Circuit flow setup, modification and teardown (controller → datapath) */
struct ofp_cflow_mod {
    struct ofp_header header;
    struct ofp_connect connect;

    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_action_header actions[0];
};
OFP_ASSERT(sizeof(struct ofp_cflow_mod) == 24);
```

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 previous 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. For a regular ADD, MODIFY_STRICT or DELETE_STRICT no associated action is currently required as all information is contained within the ofp_connect struct. We expect this to change as we develop the protocol.

3.5 Circuit Switch Features and Configuration

3.5.1 Circuit 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 */
};

```

3.5.2 Circuit Switch Configuration

The controller is able to set and query configuration parameters in the switch with the OFPT_SET_CONFIG and OFPT_GET_CONFIG_REQUEST messages, respectively. The switch responds to a configuration request with an OFPT_GET_CONFIG_REPLY message; it does not reply to a request to set the configuration. There is no body for OFPT_GET_CONFIG_REQUEST beyond the OpenFlow header. The OFPT_SET_CONFIG and OFPT_GET_CONFIG_REPLY use the following:

```

/* Circuit Switch configuration */
struct ofp_cswitch_config {
    struct ofp_header header;
    uint64_t datapath_id;      /* Datapath unique ID. Only the lower 48-bits are meaningful. */

    uint16_t flags;           /* OFPC_* flags */
    uint32_t capabilities;    /* Bitmap of supported "ofp_capabilities". */
    uint32_t actions;        /* Bitmap of supported "ofp_action_type"s. */
    uint8_t pad[2];
};
OFP_ASSERT(sizeof(struct ofp_cswitch_config) == 28);

```

The configuration flags include the following:

```

enum ofp_config_flags {
    /* Tells datapath to notify the controller of expired flow entries. */
    OFPC_SEND_FLOW_EXP = 1 << 0,
    /* Handling of IP fragments. */
    OFPC_FRAG_NORMAL = 0 << 1, /* No special handling for fragments. */
    OFPC_FRAG_DROP = 1 << 1, /* Drop fragments. */
    OFPC_FRAG_REASM = 2 << 1, /* Reassemble (only if OFPC_IP_REASM set). */
    OFPC_FRAG_MASK = 3 << 1
};

```

The flags used are the same as in the packet switch specification v0.8.9. Only the SEND_FLOW_EXP flag has meaning in a circuit switch, when a set hard_timeout expires for a circuit flow.