Packet type-aware pipeline (EXT-112) and Generic Encap/Decap (EXT-382) in OVS for the support of MPLS over GRE and NSH encapsulation

Revision History

Rev	Date	Author	Comments
0.1		Jan Scheurich	Initial version
0.2		Zoltan Balogh	Updated after meeting 2016-11-17
0.3	2016-11-29	Jan Scheurich	Updated after meeting 2016-11-23 - Revised PTAP chapters - Added EXT-382 chapter
0.4	2016-11-30	Jan Scheurich	Updated after meeting 2016-11-29 - Revised the EXT-382 chapter
0.5	2016-12-07	Jan Scheurich	Added chapter on NSH Support
0.6	2016-12-13	Jan Scheurich	Updated after meeting 2016-12-07 - Resolved questions - Added NSH encap/decap
0.7	2016-12-18	Jan Scheurich	Added new NSH field nsh_flags OXM class 0x8004 reserved for NSH First draft of work-split
0.8	2016-12-22	Jan Scheurich	Updated after meeting Dec 21

Background

Packet type-aware Pipeline (PTAP, EXT-112)

OpenFlow up to version 1.4 only supports L2 (Ethernet) packets. At any point in time a packet in the OF pipeline must have an Ethernet header. Particular push/pop actions were defined to insert/delete certain "tunnel" headers within the packet (VLAN, MPLS, PBB).

OpenFlow 1.5 standardized the "Packet type-aware pipeline" (EXT-112), which allows an OpenFlow switch to handle non-Ethernet packets. It introduces the new match field packet_type encoding the type of packet and new pre-requisites for existing match fields.

L3 Tunneling

Traditionally OVS implements tunnels as logical OF ports. Supported tunnel types include GRE, VXLAN, GENEVE, STT. As the datapath and OF pipeline in OVS are Ethernet-only, all tunnel ports are L2 tunnels, even though some of the tunnel protocols would support many other types of payload (GRE, GENEVE or VXLAN-GPE). We call these “versatile” tunnels.

The need in OVS for non-L2 tunnels such as MPLS in GRE or LISP in order to interwork with external network equipment has driven the development of a series of patches known as "L3 Tunneling support" by Lorand Jakab, Thomas Morin, Simon Horman and lately Jiri Benc. Tunnel ports that carry non-L2 payload are marked with an ”l3=true” option.
In essence these patches make OVS pop the Ethernet header when sending a packet to an "L3" port and to push a dummy Ethernet header with the correct Ethertype when receiving a packet from an "L3" port. The ofproto-dpif layer transparently adds internal push/pop_eth actions in the datapath for these transformations when needed. These are invisible for the OpenFlow controller.

Jiri’s patches for the kernel datapath have recently been upstreamed in Linux net-next. In the kernel datapath a new u8 mac_proto member was added to sw_flow_key structure to distinguish L2 and L3 packets. The presence of netlink key attribute OVS_KEY_ATTR_ETHERNET is used to indicate if it’s about L2 or L3 packet on the netlink interface. The “L3” packet type is encoded in the OVS_KEY_ATTR_ETHERTYPE netlink attribute.

The latest user-space patches are not yet upstreamed and we suggest to revise them in the context of the present proposal. They are doing far more than necessary to stitch L3 tunnels to OVS’ L2 pipeline but not enough to go all the way for OVS to support PTAP.

NSH

NSH is a service tunnel header that can encapsulate different packet types, typically Ethernet or IP. For transport it requires an outer Ethernet header or a generic transport tunnel such as VXLAN-GPE or GRE that can carry NSH frames.

The work on NSH patches for OVS work is stuck mainly because of the difficulty to handle the variable encapsulations. For VXLAN-GPE transport it depends on the L3-Tunneling support. Furthermore the OpenFlow pipeline must be able to push/pop an NSH header both with and without Ethernet header. This calls for separate actions to push/pop NSH and push/pop Ethernet headers. This implies the capability of OVS to handle non-Ethernet packets in the OF pipeline, i.e. PTAP.

As an extension to PTAP ONF is about to standardize in ETX-382 generic encap/decap actions to add and remove arbitrary tunnel encapsulation headers at the front of a packet and thus change the type of the packet in the pipeline. These appear to be the right tools to push and pop NSH and Ethernet headers individually.

Proposed Technical Solution

PTAP as optional capability of a bridge

We plan to implement the PTAP as specified in OF 1.5.1, supporting the new packet_type match field with the following header type namespaces:

- 0 (OFPHTN_ONF)
- 1 (OFPHTN_ETHERTYPE)

Initially we should support the following packet types:

- (0, 0) Ethernet,
- (1, 0x800) IPv4,
- (1, 0x86dd) IPv6,
- (1, 0x8847) MPLS,

as these are the natural payload of L3 tunnels we want to support. Adding other packet types shall be possible and easy. Packet_in and Packet_out will be supported for these packet types.

Support of PTAP should be a configurable option of a bridge. Even though ONF has taken some care to minimize the impact on controllers when controlling a PTAP switch that only handles Ethernet packets, a controller should be aware of the PTAP changes when dealing with a PTAP switch.

To guarantee backward-compatibility the default mode of an OVS bridge should be non-PTAP (or legacy). In this mode the pipeline does not support the packet_type match field and handles only L2 packets. On the OpenFlow interface OVS must behave in all aspects as if PTAP was not implemented.

Interwork Between Tunnel Ports and OVS Bridges

With PTAP in place it is natural to let "versatile" tunnel ports (tunnels which support multiple payloads, e.g. GRE, GENEVE, VXLAN-GPE) directly exchange their payload packets with the PTAP pipeline, mapping the tunnel's protocol field to packet_type and vice versa and dropping packets where no such mapping is defined. A single versatile tunnel port could thus carry Ethernet and non-Ethernet packets and there would be no need for a tunnel port option to specify the behavior. The PTAP OF controller would have full responsibility of matching on received packet type and setting the desired packet type before sending a packet to a versatile tunnel port, using the generic encap/decap actions introduced with EXT-382 (see below).

For legacy, non-PTAP bridges, the pipeline processing will remain strictly limited to Ethernet packets. To let non-PTAP bridges interwork with versatile tunnels we propose to keep the concept of an optional “Layer3” tunnel mode option introduced as part of the current L3 tunneling patch set. Tunnel ports without that option behave like legacy layer 2 tunnels and carry the Ethernet frame of the non-PTAP pipeline unchanged.

For tunnel ports with the “Layer3” option set, a non-PTAP bridge pushes a dummy Ethernet header immediately at reception of a non-Ethernet packet from the L3 tunnel port and pops the L2 header directly before transmission to an L3 tunnel port. We suggest that the bridge also pops a VLAN tag in the L2 header, if any.

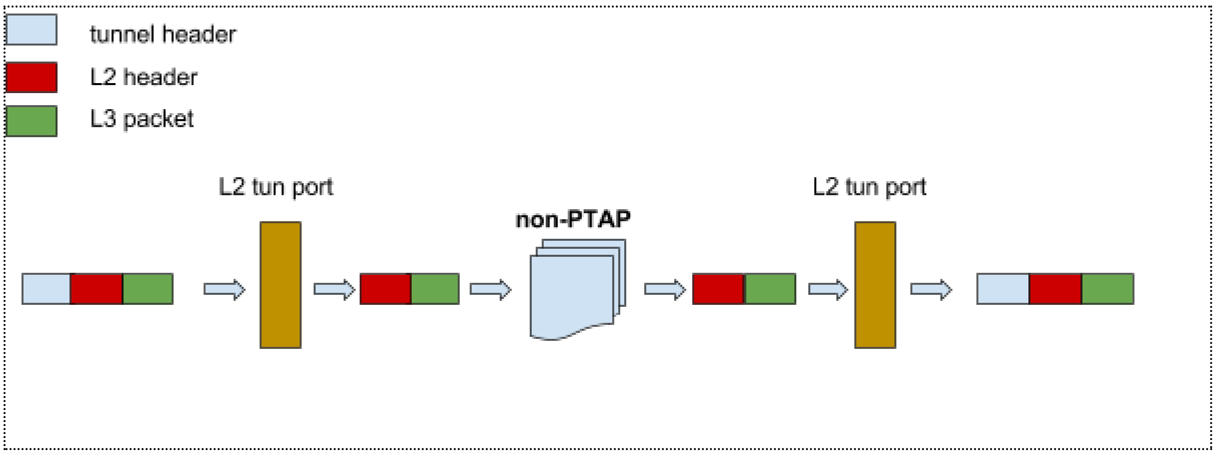
The “Layer3” option is only applicable for tunnel ports connected to a non-PTAP bridge. It should be rejected when the tunnel port is added to a PTAP bridge.

The interwork between tunnel port and bridge modes is summarized in the tables below

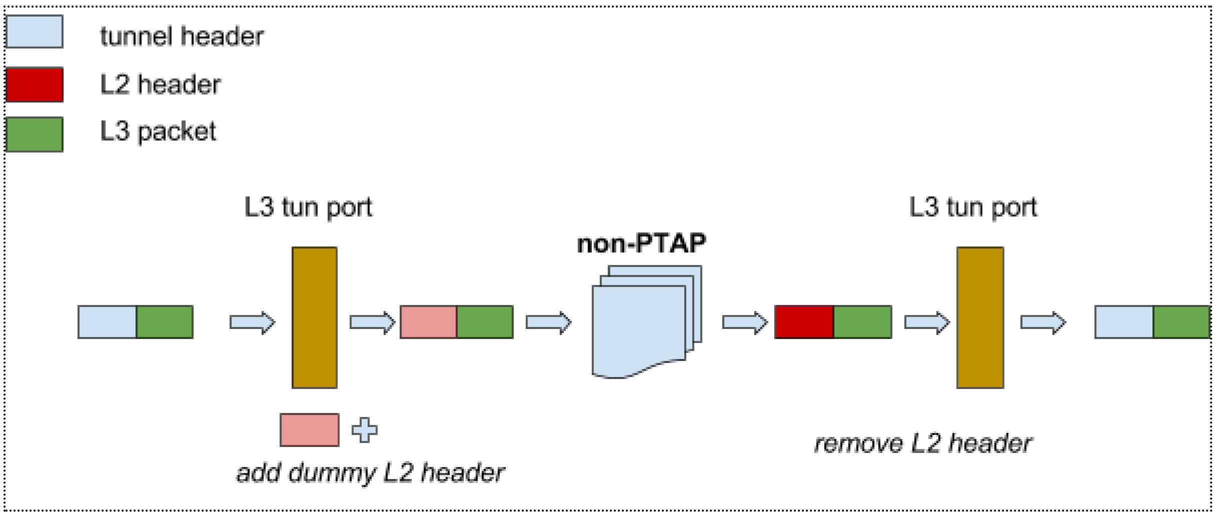
Receive from	Default mode	Layer3 mode
non-PTAP bridge	Receive Ethernet packet as is, Drop non-Ethernet packets	Push dummy Ethernet header, Drop Ethernet packets
PTAP bridge	Receive any packet with supported type as is, else drop packet	N/A

Transmit to	Default mode	Layer3 mode
non-PTAP bridge	Send Ethernet packet	Pop L2 header, send if tunnel supports packet type, else drop
PTAP bridge	Send any packet as is if tunnel supports packet type, else drop.	N/A

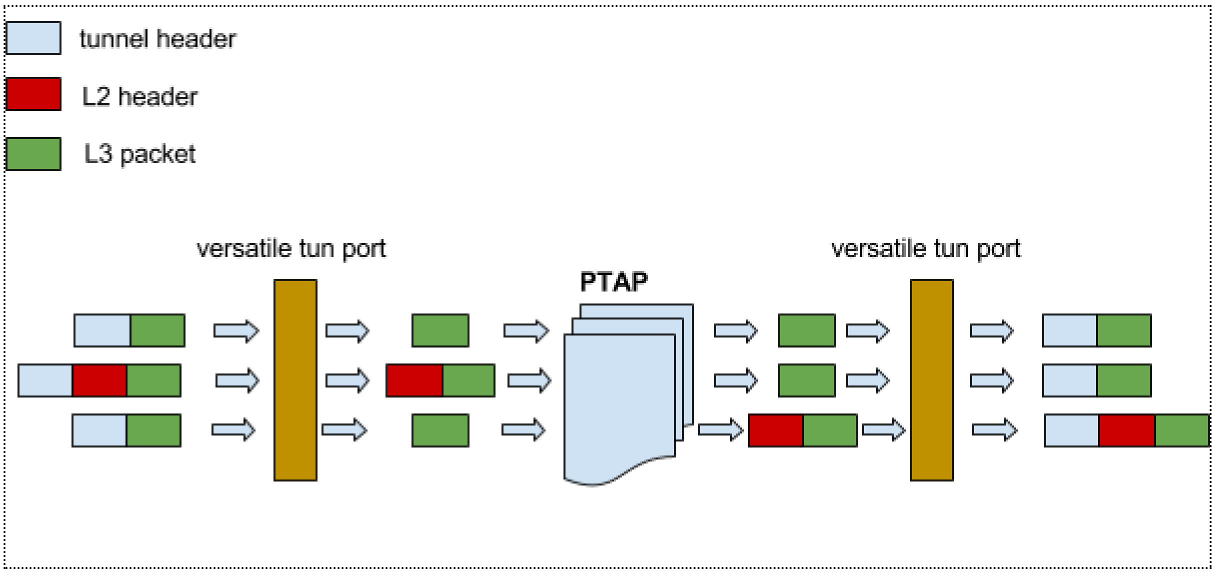
The figures below illustrate these cases.



Default (L2) tunnel port with non-PTAP bridge



“Layer3” tunnel port with non-PTAP bridge



Full flexibility of PTAP bridge with versatile tunnel ports

Compatibility Considerations

A tunnel port in default mode behaves the same when connected to a non-PTAP and a PTAP bridge with a legacy OpenFlow pipeline as long as the tunnel carries Ethernet packets. Non-Ethernet packets would be dropped by the non-PTAP bridge but processed by the PTAP bridge. Since the received packets do not match the implicit Ethernet packet type, the packets will not match any table entries.

A tunnel port in “Layer3” mode behaves differently when connected to a PTAP bridge compared to a non-PTAP bridge with the same legacy OpenFlow pipeline because the received packet type will be non-Ethernet but the flow entries implicitly match on packet type Ethernet, resulting in table misses. So controllers must typically be updated to make use of PTAP pipelines for Layer3 tunnels.

A simple (albeit non-efficient) way for a controller to maintain compatibility of existing OF pipelines would be to explicitly push a dummy Ethernet header when receiving non-ethernet packets from a tunnel port.

Support of Multiple Packet Types in one Bridge

The OF 1.5.1 spec officially only supports one packet type per switch, mainly because there is no way of the switch to modify the packet type. The implementation in OVS must support multiple packet types per bridge for interwork between L2 and L3 tunnels and to allow modification of the packet type through EXT-382.

For OpenFlow tables that match on L3/L4 fields only, this means there are now two equivalent but distinct matches. For matching on an IPv4 destination IP address these are

1. packet_type=(0, 0), ethertype=0x800, nw_dst
2. packet_type=(1, 0x800), nw_dst

To avoid duplication of all such flow entries in case the pipeline can be fed with different packet types, the controller could decap a present Ethernet header to normalize packet type before sending the packet to the L3 flow tables and subsequently encap a new Ethernet header when sending the packet to an Ethernet port.

Note: This is a logical decapsulation only. In the OVS datapath this can be optimized as to avoid any unnecessary overhead due to decap and subsequent re-encapsulation.

Discovery of packet types supported by a Bridge or tunnel port

According to the OF 1.5.1 spec [7], the switch must include the list of all supported packet types per OF table in a new OFPTFPT_PACKET_TYPES property in Table Features Reply message (page 152) if it supports other than the Ethernet packet type (0, 0).

In OVS every OF table of a PTAP bridge would support the complete set of packet types. The Table Features Reply needs to be enhanced to list all supported packet types for every table. Every time support for a new packet type is being added, the returned list must be updated.

In principle a controller should query the Table Features of a switch to find out if it supports multiple packet types.

For tunnel ports attached to a PTAP bridge, the controller also needs to know which packet types are supported.

Open Questions on PTAP and Tunnel Port Interaction

Q: Is there a simpler way for a controller to check the support of packet_types in a switch than querying the Table Features? A controller with OVSDB interface could query the bridge options instead.

Q: How can PTAP controller find out what packet types can be sent to/received from any port, in particular for “versatile” tunnel ports? Do we need new OF port properties for that?

Q: Shall the switch drop non-Ethernet packets that a PTAP bridge tries to send to an Ethernet-only port?

Q: Should PTAP be limited to OF1.5 or can we also make the packet_type match field an Nicira extension to OF1.3/1.4 in OVS?

Q: Is a PTAP bridge compatible with the “NORMAL” MAC-learning mode? Probably not as packet’s are not guaranteed to be L2.

Q: Should patch ports between PTAP and non-PTAP bridges be allowed? If so, what should their behavior be?
The use case we have in mind is an OF-controlled PTAP integration bridge having both L3 tunnels as well as VLAN-tagged patch ports over a “physical” bridge operating in NORMAL mode, where the latter can’t operate in PTAP mode (see Q above).
One possibility could be to define patch ports as Ethernet-only and hold the PTAP controller responsible to only send Ethernet packets to a patch port.

Resolved Questions on PTAP and Tunnel Port Interaction

Q: Should we support tunnel port modes “L2” and “L3” in connection with PTAP bridges?
A: No. PTAP bridges will support only versatile ports.

Q: Should we support tunnel port mode “versatile” with non-PTAP bridge?
A: No. Non-PTAP bridges will support only L2 and L3 ports.

Q: How to handle L3 packet received on L2 tunnel port in a non-PTAP bridge?
A: Non-Ethernet packets (including L3) received on L2 tunnel port should be dropped.

Q: How to handle L2 packet received on L3 tunnel port in a non-PTAP bridge?
A: Ethernet frames received on L3 tunnel port should be dropped.

Q: Is the proposal of PTAP/non-PTAP bridges aligned with the kernel patch?
A: Yes, it is. The kernel datapath with Jiri’s kernel patch handles all tunnel ports as versatile. Any L2/L3 logic for legacy non-PTAP pipeline must be handled in ofproto layer.

Q: How to configure L3 interfaces in PTAP bridges? Would a boolean option like ‘layer3’ be sufficient or should an enum be used? That way besides L2 and L3 a third ‘versatile’ mode could be configured.
A: The ‘layer3’ option will be used only in non-PTAP bridges to define L3/L2 interfaces. All the ports in PTAP bridges will be versatile, so there is no reason to use the ‘layer3’ option.

Q: Should the implicit push/pop_eth actions required by a non-PTAP bridge be handled by the vport directly without involvement of datapath and ofproto or rather be inserted by ofproto-dpif?
A: ofproto-dpif seems more logical because otherwise the vport behaviour would have to depend on the type of bridge.

Generic Tunnel Encap/Decap Actions (EXT-382)

Ext-382 proposes two new generic OF actions to add/remove tunnel encapsulation header to/from the front of a packet in the OF pipeline and thus change the type of the packet.

```
/* Action  structure for OFPAT_ENCAP */
struct ofp_action_encap {
        uint16_t       type;          /* OFPAT_ENCAP */
        uint16_t       len;           /* Total size including any property TLVs */
        struct ofp_header_type new_pkt_type;
                                      /* Header type to add and PACKET_TYPE of result */
        uint16_t       hdr_size;      /* (optional) Header size in bytes to add,
                                         0=not specified*/
        uint8_t pad[6];               /* Align to 64bits. */
        struct ofp_ed_prop_header props[0]; /* encap/decap properties */
};
```

New_packet_type indicates both the header type to push and the packet type of the resulting packet. The encoding should be aligned with the packet_type match field in the specification.

In the following we limit the discussion to use cases that push an individual packet header whose layout is understood by the switch and which it can parse, match on and manipulate when present in a packet.

Fixed header fields for which there are match fields defined should be initialized to zero and can be modified with set_field ot copy_field actions after the encap action. Fixed header fields not specified as match fields (e.g. crc or length) must be set appropriately by the encap action to create a well-formed packet header.

If the header contains some kind of “next protocol” field, the encap action should, if possible, set it in accordance with the packet type of the encapsulated packet. If the “next protocol” field is specified as a match field, the controller can override the default value with a subsequent set_field action.

Other header fields (such as optional or variable length fields) should be set based on ofp_ed_prop_header property fields included in the encap action.

Under normal circumstances the hdr_size parameter should not be used if the switch understands the encap header as the switch can derive the total header size from the action and any included properties. In OVS we are not going to support a non-zero hdr_size parameter for the encapsulation we introduce in this context.

The generic decap action removes the outermost header from the packet

```
/* Action  structure for OFPAT_DECAP */
struct ofp_action_decap {
        uint16_t       type;          /* OFPAT_DECAP */
        uint16_t       len;           /* Total size including any property TLVs */
        struct ofp_header_type new_pkt_type;        /* PACKET_TYPE of result */
```

```
       uint16_t       hdr_size;      /* (optional) Header size in bytes to remove,
                                         0=not specified*/
       uint8_tpad[6];        /* Align to 64bits. */
       struct ofp_ed_prop_header props[0]; /* encap/decap properties */
};
```

The header to be removed is implicitly determined by the current packet_type of the packet. The new_pkt_type field determines the packet_type of the resulting decapsulated packet. The special value (0, OFPHTO_USE_NEXT_PROTO) can be used to let the switch set the resulting packet type based on the current header's "next protocol" field.

The hdr_size parameter can be used to tell the switch how many bytes to pop in case the switch cannot determine the size of the outer header itself. In OVS we are not going to support a non-zero hdr_size for decap action.

The abstract ofp_ed_prop_header TLVs are defined as follows. Each property is tagged by class and type and followed by up to 251 octets of variable length data.:

```
/* Common header for all  Encap/Decap Action  Properties */
struct ofp_ed_prop_header {
    uint16_t prop_class;      /* encap/decap property class  */
    uint8_t type;             /* encap/decap property type  */
    uint8_t len;              /* property length */
};

enum ofp_ed_prop_class {
    OFPPPC_BASIC      = 0,           /* ONF Basic class  */
    OFPPPC_MPLS       = 1,           /* MPLS property  class */
    OFPPPC_GRE = 2,           /* GRE property  class */
    OFPPPC_GTP        = 3,           /* GTP property  class */
    OFPPPC_NSH        = 4,           /* NSH property  class */
    /*  new values go here  */
    OFPPPC_EXPERIMENTER=0xffff, /* experimenter property  class
                                  * first  32 bits of property data is exp id
                                  * after  that is the experimenter property  data
                                  */
}

enum ofp_ed_basic_prop_type {
    OFPPPT_PROP_NONE =  0,           /* unused */
    OFPPPT_PROP_PORT =  1,           /* Port properties associated  with the encap/decap  */
    /*  new values go here  */
};
```

Specific of_ed_properties of type port (1) have been defined in EXT-382 to allow the implicit creation of logical (tunnel) ports through encap and decap actions. This mechanism is relatively unclear (to us) and we suggest to omit that part of EXT-382 in the first step.

Generic Encap/Decap Use Case: Ethernet

We will discuss this as the most basic example for conversion between packets of type (0,0) and (1, Ethertype) needed in a PTAP bridge to switch between L2 and L3 ports.

As Ethernet is traditionally at the heart of OpenFlow, the OpenFlow spec should define a decent support for pushing/popping Ethernet headers. Below is a proposal for specification in ONF and implementation in OVS.

The encap action for Ethernet should take the following parameters:

```
len = 16;                  /* No TLV properties */
new_pkt_type = (0, 0)      /* OFPHTN_ONF / OFPHTO_ETHERNET */
hdr_size = 0;              /* unspecified */
```

The encap action adds a 14 byte Ethernet header to the packet.

If applied to a packet of packet type (OFPHTN_ETHERTYPE, <Ethertype>), the resulting packet type is (OFPHTN_ONF / OFPHTO_ETHERNET) and the Ethertype field of the new header is by default set to the Ethertype from the previous packet type.

It makes sense and it should be possible to apply the encap(Ethernet) action also to a packet of type (OFPHTN_ONF / OFPHTO_ETHERNET). The resulting packet would be raw MAC in MAC encapsulation with unchanged packet type. The new (outer) Ethertype would by default be set to 0x6558 (Transparent Ethernet Bridging). However, as the OVS kernel datapath datapath currently does not support this operation, we will for now not allow it.

Applying the Ethernet encap action to other packet types (such as name spaces IP_PROTO or UDP_TCP_PORT) is not meaningful and would result in malformed packets.

Where OVS can statically check the packet type pre-requisite (e.g. apply_actions instruction in flow entries), it should return an error for unsupported packet types. For other cases (write_actions instructions and group buckets) OVS needs to handle it as a run-time error at action execution. Proposed error handling is to drop the packet.

Source and destination MAC address are initialized to zero and should be populated by the controller with set_field or copy_field actions after the encap action. The controller can also override the default Ethertype with a set_field action, if wanted.

The generic decap action executed on an Ethernet packet (packet type OFPHTN_ONF / OFPHTO_ETHERNET) can be parameterized independently in two dimensions:

1. Scope of the Ethernet decap action:

   In its basic form, without any additional TLV property, the decap action removes the 14 byte Ethernet header. This is only possible if the packet does not contain any VLAN tags. The controller can also avoid this situation by popping any VLAN tags before decap.

   OVS cannot statically check this prerequisite, so we need some run-time error handling at action execution. The proposed error handling is to drop the packet.

2. Determination of the resulting packet type:

   If new_pkt_type is set to (OFPHTN_ONF / OFPHTO_USE_NEXT_PROTO) the switch automatically derives the new packet type from the Ethertype of the removed header as follows: (OFPHTN_ETHERTYPE, <Ethertype>).

   As OVS has already parsed the L3 packet before there is no need to reparse the packet for further processing. Only the type of the packet changes. This should be a lightweight operation.

   Otherwise the new packet type is set to the value included in the decap action independent of the Ethertype in the removed header. In such a case the resulting packet should probably be re-parsed based on the new packet type.

Generic Encap/Decap Use Case: NSH

This will be covered in the later chapter on NSH support.

Open Questions on EXT-382

Q: Should generic encap/decap actions be allowed in OpenFlow Action Set? If so, what would be there position in the order of actions to be executed? Should two encap or decap actions be allowed if they have different type?
A: For now OVS will not support encap/decap actions in the Write Actions instruction to avoid these complications. Using them in Apply Actions in flow entries gives full flexibility. However, it would be desirable to be able to use encap actions also in group buckets, which conceptually also contain Action Sets, e.g. to implement tunnels.

Resolved Question on EXT-382

Q: What is the use case of specifying hdr_size != zero in encap? If the switch does not know or cannot determine the total size of the header to be pushed from the action and its included ofp_ed_prop_headers, it would have to leave part of the added header space uninitialized (or zeroed), which hardly represents a valid packet. If the switch does not know the header, the only reasonable use case would be for the controller to provide the entire header as a binary chunk in form of one (or more) TLV ofp_ed_prop_header properties. As these already include a length, specifying hdr_size in addition would be redundant.
A: We still cannot see a real use-case for this and suggest that ONF re-consider before freezing the encap action in OF 1.6.

Q: Should new_pkt_type be uint32_t or struct ofp_header_type (16 bit ns, 16 bit ns_type)?
A: The new_pkt_type in encap/decap will be struct ofp_header_type. But the packet_type match field is a non-maskable 32 bit integer in OF 1.5 where the upper 16 bits are ns and the lower 16 bits are the ns_type. In OVS we suggest to make this field maskable.

Q: Should non-canonical packet types be allowed as new_pkt_type in encap?

A: Only canonical packet_types should be supported, as is required for further processing in the pipeline. The actual packet type (if there are multiple possible values) is only relevant when another encap is to be added. In that case the controller could use the true packet_type to populate the “next protocol” field of the outer header.

Q: EXT-382 states that the decap action assumes the expected header is present and the behaviour of decap is undefined if not. As far as I can see the decap action does not have an expected packet type.
A: That statement is a leftover and should be removed from the EXT-382 specification.

Q: How does ONF foresee the specification of ofp_ed_properties and the rules for their use in encap and/or decap actions for specific header types?
A: The idea is that the definition of specific ofp_ed_properties for generic encap and decap actions can be done outside the core OpenFlow specification, e.g. in technology or protocol-specific supplements. For a few core encap/decap use cases, e.g. Ethernet, the behaviour and basic ed_properties should go into the OF spec.

Q: Should encap(Ethernet) action be allowed for Ethernet packets? If so, what is the expected result? Raw MAC in MAC encapsulation? Which Ethertype?
A: This should be allowed and result in raw MAC in MAC encapsulation. A possible default Ethertype could be TEB (0x6558), which the controller could overwrite with set_field.

Q: Should the decap action when applied on an Ethernet packet also pop any present VLAN tag(s) and leave the L3 packet? Or should it just pop the Ethernet header and leave a VLAN packet of type e.g. (OFPHTN_ETHERTYPE, 0x8100)?
A: OVS will not support decap on Ethernet packets that have VLAN tags. The controller must pop any VLANs before decap to avoid this problem.

Q: Should the match fields corresponding to the decapsulated header be cleared immediately at decap or just before the next table match (recirculation)? Is there a compelling use case that would suggest keeping fields during action list processing (e.g. [decap, manipulate, encap])?
A: It is conceptually difficult to attach a meaning to header match fields after decap. There might be another instance of the same header following; do the fields now refer to the removed or the inner header? To avoid such problems match fields related to the removed header are lost with the decap action and the revealed inner packet is re-parsed (if necessary). A controller can copy headers to metadata (e.g. registers), if needed.
.
Q: Should the generic encap/decap OF actions be mapped to specific push/pop_<header> actions on the DPIF interface? We already have the push/pop_eth actions for L3 tunnel support with non-PTAP bridges. Is there any advantage to go for generic actions on DPIF?
A: For now, we should stick to specific dedicated push/pop_<header> actions on DPIF, mainly to ease the interwork with the Linux kernel datapath. On the interface to the netdev datapath we have more freedom but (for reasons of code complexity and maintainability) we should only let translation diverge between both datapaths if there are very good reasons. In the long run the kernel datapath might also become more flexible with the introduction of eBPF.

Support for NSH

NSH Protocol

The NSH draft defines two variants of the NSH header. A fixed size header (MD Type = 1) and a variable size header (MD Type = 2) where the NSH metadata fields are encoded as a list of TLV headers.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver|O|C|R|R|R|R|R|R| Length=6  | MD type=0x1   | Next Protocol |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Service Path Identifier (SPI) | Service Index (SI)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Mandatory Context Header 1                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Mandatory Context Header 2                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Mandatory Context Header 3                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Mandatory Context Header 4                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Network Service Header MD Type 1

For MD 1 The version field (Ver) and the C flag must be set to zero. The O flag must only be set for NSH OaM packets. The NSH draft specifies the following Next Protocol values:

0x1 : IPv4
0x2 : IPv6
0x3 : Ethernet
0x4: NSH
0x5: MPLS
0xFE-0xFF: Experimental

The base NSH draft does not specify how the mandatory context headers are to be populated. This is left to other specifications or service function chaining implementations.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver|O|C|R|R|R|R|R|R| Length    | MD type=0x2   | Next Protocol |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Service Path Identifier (SPI) | Service Index (SI)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~ Variable Length Context Headers (opt.)                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Figure 2: Network Service Header MD Type 2

Also here Ver must be set to zero. The C flag must be set if the header contains any critical variable length context headers.

Each variable length MD 2 context header is TLV-encoded as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Metadata Class                |C| Type        |R| Len         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Variable Metadata                                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class and Type form a hierarchical name space for MD2 headers. The C flag marks a header as critical. An SFC node that doesn't understand a critical TLV must drop the packet. The C flag is considered to be part of the 8-bit type field, so that critical TLVs have type values in the range 128-255.

This TLV encoding differs from the encoding of optional headers in Geneve only by the fact that the Len field specifies the total header value length in **bytes**. A TLV header that does not occupy an integer multiple of 4-byte words must be padded to a 4-byte boundary by the sender but padding octets are not included in the Len value. In Geneve the Len field contains the length in entire 4-byte words (including any possible padding octets defined for a specific TLV type).

NSH Transport

NSH packets must be encapsulated for transport. They can be carried directly on Ethernet (Ethertype 0x894F) or on versatile transport tunnel protocols like GRE, VXLAN-GPE or Geneve. Currently the most frequently discussed tunnel protocol is VXLAN-GPE.

Basic NSH Use Cases

The SFC standards foresee a number of basic use cases from which we can derive the following scenarios for an SFC switch dealing NSH encapsulation

SFC Classifier

This is a prioritized use case for OVS.

A (not further specified) classification of a packet results in the selection of a Rendered Service Path (RSP) identified by a Service Path ID (SPI). The classifier pushes an NSH header onto the packet, initializes the SPI to the chosen RSP and the Service Index (SI) to the initial value configured for the RSP (default 255). The SFC may also add context headers to the pushed NSH header, either in the mandatory context header fields (MD1) or as TLV context headers (MD2).

The classifier then forwards the encapsulated packet to the first SFF in the RSP. This SFF may be co-located on the same switch, in which case the NSH packet is handed over internally to the SFF function. If the first SFF is not co-located, the classifier forwards the NSH packet over a suitable transport (e.g. a VXLAN-GPE) tunnel to the remote SFF.

Note: In the current SFC demonstrations classifiers typically encapsulate the entire L2 packet in the NSH header (Next protocol = 3), thus preserving the original Ethernet header throughout the service chain. In principle this would allow L2 forwarding at the end of the service chain. Whether that is a realistic use case or just coincidence because OVS as classifier works on L2 frames is unclear. Given that SFC would normally be applied to IP traffic it would also make sense to encapsulate the IP packets in the NSH header and rely on L3 forwarding at the end of the chain.

Both options shall be supported, which means that it is up to the SFC entities that are exposed to inner packets (final SFF, Service Functions and SFC proxies) to deal with L2 and/or L3 packets properly.

Service Function Forwarder (SFF)

This is a prioritized use case for OVS.

The SFF receives an NSH encapsulated packet from a co-located classifier or remotely over some transport channel. The SFF inspects the SPI and SI fields of the NSH to determine the SFC next hop, typically a Service Function (SF).

In advanced use cases the SFF may also consider other context headers for the forwarding decision: For autonomous load-balancing between multiple SF instances in an SFF, the NSH header can for example carry a hash of the inner packet for consistent path selection.

The SFF then forwards the unchanged NSH packet to the SF over some transport channel. The SF can be locally connected to the SFF, in which case the NSH packet may be sent over Ethernet, in other cases the transport to the SF may be another transport tunnel.

It is the SF’s responsibility to return the processed packet to the SFF with the SI decremented by one. In advanced use cases the SF may reclassify the packet and overwrite the SPI to select a new service path.

When the SFF receives the returned packet it forwards it to the next hop based on the new combination of SPI and SI. There are three possible forwarding cases:

a Another SF reachable from the same SFF
b The next SFF in the path
c The end of the service chain

Cases a and b basically are similar to the cases described above. In case c, the SFF removes the NSH header to expose the inner packet and forwards it onward to its final destination. Depending on the Next protocol the packet would be L2- or L3 forwarded. Typically the SFF derives the applicable forwarding context (L2 bridge or L3 VRF) from some NSH context header that was inserted by the classifier. For this the final SFF must be able to save some NSH context headers in packet metadata before the decap operation to be able to use it as input for further forwarding.

SFC Proxy

The SFC proxy resides between an SFF and a non NSH-aware SF. Its role is to remove the NSH header from packets sent to the non-NSH SF and re-attach the NSH header (with decremented SI) to the packet when it returns. The SFC proxy function is conceptually simple but very hard to implement without detailed knowledge of how the SF processes (and potentially modifies) packets.

Unless it re-implements the full original classifier, an NSH proxy must be stateful in order to store the popped NSH headers. Even with storage capabilities, the SFC proxy must be able to uniquely associate packets returned by the SF with their stored NSH headers. This can be arbitrarily hard if a SF modifies packets. A certain class of SFs that do not modify L3/L4 headers can be handled by tracking L4 connections. But a fully generic stateful SFC proxy seems infeasible.

A stateful SFC proxy does not appear a natural function for implementation with OpenFlow in OVS. A stateless SFC proxy implemented through re-classification can be decomposed into the decapsulation and classifier functions as described above, so it does not add new requirements on OVS.

Service Function

A SF may read any part of the NSH header but also modify it in many ways. At minimum it must decrement the SI field. Service functions operate on the inner packets so they need to parse/manipulate the inner packet while preserving the NSH header.

For OVS to implement an NSH-aware service function, it would need to save the entire decapped NSH header as some form of packet metadata and reconstruct the NSH header from the saved metadata at the end of processing of the inner packet.

This is not a prioritized NSH use case for OVS.

Advanced NSH Use Cases

Hierarchical Service Chaining

IETF Draft [draft-ietf-sfc-hierarchical] describes a hierarchical architecture for service chaining. In in so-called IBN nodes at the boundary between hierarchy layers decide on the service path segment in the lower layer. To do that they attach an outer NSH to the packet, which is used during the traversal of the lower segment and removed at the end of the segment to reveal the inner NSH of the higher layer. This implies NSH in NSH encapsulation.

To be able to address such use cases with OVS in the future, OVS must support stacks of more than one NSH header.

Parsing NSH Packets and Matching on NSH Headers

OVS may receive NSH packets in two flavours:

1. Packet type (0,0), Ethertype 0x894F	from Ethernet ports
2. Packet type (1,0x894F)	from versatile tunnels in PTAP bridge

In both cases OVS should parse the NSH header (including any TLVs in MD2 format) and make the NSH header fields available for matching and in-place manipulation through set_field and copy_field actions.

In analogy to what was discussed above for IP packets, the controller should normalize NSH packets to one packet type, e.g. (1,0x894F), before submitting them to NSH processing to avoid duplication of flow entries and problems and cumbersome case handling before transmit to ports/tunnels.

ONF have allocated the OXM class 0x8004 for NSH match fields (see [ONF Registry]). The NSH match fields can therefore be defined as OXM fields.

All NSH fields share the two alternative pre-requisites specified above.

NSH Header field	OXM code	ovs-ofctl field name	Size [bits]	Comments
Flags	OXM_NSH_FLAGS (0x8004,1)	nsh_flags	8	Bits 2-9 of first NSH word
MD Type	OXM_NSH_MDTYPE (0x8004,2)	nsh_mdtype	8	Bit 16-23
Next Protocol	OXM_NSH_NEXTPROTO (0x8004,3)	nsh_np	8	Bits 24-31
SPI	OXM_NSH_SPI (0x8004,4)	nsh_spi	24	Bits 0-23 of second NSH word
SI	OXM_NSH_SI (0x8004,5)	nsh_si	8	Bits 24-31
Context Header 1	OXM_NSH_C1 (0x8004,6)	nsh_c1	32	Prerequisite nsh_mdtype == 1
Context Header 2	OXM_NSH_C2 (0x8004,7)	nsh_c2	32	Prerequisite nsh_mdtype == 1
Context Header 3	OXM_NSH_C3	nsh_c3	32	Prerequisite

	(0x8004,8)			nsh_mdtype == 1
Context Header 4	OXM_NSH_C4 (0x8004,9)	nsh_c4	32	Prerequisite nsh_mdtype == 1
Variable length context headers	OXM_GEN_TLV<N> (0x8005, <N>) N = 0 … 63	gen_tlv<N> N = 0 … 63	Var length up to 128 bytes	

Handling of MD2 TLV headers

The original proposal for MD2 TLV headers was to re-use the tunnel metadata infrastructure originally implemented for Geneve tunnels. It appears cleaner to differentiate these conceptually different types of match fields (tunnel metadata vs. packet headers) on OpenFlow level. Internally these can still be mapped to the a common set of flow registers, so that gen_tlv<X> is effectively an alias for tun_metadata<X>.

Optimally the new gen_tlv<X> register fields should be assigned a dedicated NXM class to be able to use the suffix <X> as field id directly.

Q: The standard 64-bit registers defined in have been assigned a dedicated OXM class OFPXMC_PACKET_REGS in [Openflow 1.5.1]. Could we do similar for gen_tlv registers even though they are not yet standardized in ONF?

The actual handling in the datapath requires investigation as it seems impossible to store the raw TLVs from a Geneve tunnel and the NSH MD2 TLVs inside the same memory area in struct flow.

The existing Nicira OpenFlow commands to configure a mapping of TLVs to tun_metadata fields should be enhanced to include a protocol id to uniquely identify a TLV. Otherwise there is a risk of collisions between {class, type} pairs defined independently in two protocols. For example:

```
ovs-ofctl add-tlv-map br0 "{proto=geneve,class=0xffff,type=0,len=4}->tun_metadata0"
ovs-ofctl add-tlv-map br0 "{proto=nsh,class=0xfff6,type=10,len=4}->gen_tlv1"
```

To do this we can introduce a new Nicira extension message type (add-tlv-map1) with the proto field added. The legacy add-tlv-map command can still be used for Geneve TLV headers for backward compatibility. The ovs-ofctl command line interface will automatically pick the right message type.

The controller is responsible to not assign TLVs from different protocols to the same gen_tlv or tun_metadata register. As they are shared, the first mapping would be overwritten by the second.

Actions on NSH Headers

The existing set_field and copy_field/move actions in OVS are sufficient for almost all of the NSH use cases described above.

The only NSH-specific action we should consider adding is a **dec_nsh_si** action to decrement the NSH SI field by one as it is a very common operation on the NSH header and follows the principles for other headers with TTL fields (IP, MPLS). It cannot easily be emulated using existing OpenFlow means. However, this is not prioritized as it is not needed for the basic use cases.

Encap and Decap Actions For NSH

The generic encap action for NSH will be supported for the following inner packet types:

(0, 0)	Ethernet
(1, 0x800)	IPv4
(1, 0x86dd)	IPv6
(1, 0x8847)	MPLS
(1, 0x894f)	NSH

It take the following parameters:

```
len = 16 + Len;                    /* Depending on TLV properties */
new_pkt_type = (1, 0x894F)         /* NS OFPHTN_ETHERTYPE / NSH */
hdr_size = 0;                      /* unspecified */
```

The encap(NSH) action takes one mandatory fixed size property, determining the metadata format:

```
struct ofp_ed_prop_nsh_md_type {
    uint16_t prop_class;      /* encap/decap property class NSH = 4 */
    uint8_t type;             /* encap/decap property type MD_TYPE = 1 */
    uint8_t len;              /* property length  = 5 */
    uint8_t md_type;          /* NSH MD type */
    uint8_t pad[3];           /* Padding to 8 bytes */
};
```

Currently only the MD types 1 and 2 will be supported. If the md_type property is set to 1, the encap(NSH) action prepends a fixed header of size 24 octets to the packet and initializes the header fields with the following values:

- Ver = 0
- Flag O = 0
- Flag C = 0
- Length = 6
- MD type = 1
- Next Protocol = 1-5 (depending packet type of inner packet)
- SPI = 0
- SI = 255

- Context headers 1-4 = 0

After the encap(NSH) action, OVS has conceptually parsed the new header so that the NSH OXM fields are populated with the pushed values.

The controller can set the SPI and, if necessary, the SI value after encap with set_field actions on OXM match fields nsh_spi and nsh_si. It can further set the fixed context headers through OXM match fields nsh_c1 through nsh_c4.

To mark the NSH header as NSH OAM packet, the controller could add the following optional encap property:

```
struct ofp_ed_prop_nsh_oam {
    uint16_t prop_class;      /* encap/decap property class NSH = 4 */
    uint8_t type;             /* encap/decap property type OAM = 2 */
    uint8_t len;              /* property length  = 4 */
};
```

When present in the encap action, OVS will set the O flag to 1.

For an encap(NSH) action with md_type property “2” the controller must add the TLV context headers to be pushed in form of zero or more encapsulation properties of the following type:

```
struct ofp_ed_prop_nsh_tlv_context_hdr {
    uint16_t prop_class;      /* encap/decap property class NSH = 4 */
    uint8_t type;             /* encap/decap property type TLV_CTX_HDR = 3 */
    uint8_t len;              /* property length  = 8 + tlv_len + padding */
    uint16_t tlv_class;              /* Metadata class */
    uint8_t tlv_type;         /* Metadata type including C bit */
    uint8_t tlv_len;          /* Metadata value length (0-127) */
    uint8_t tlv_data[0];      /* tlv_len octets of metadata value,
                                 padded to a multiple of 4 bytes */
};
```

The TLV context headers are pushed to the NSH header in the same order as they are specified in the encap(NSH) action. OVS will add enough space for the NSH header to accommodate the fixed 8 octets plus all specified TLV context headers including their padding. The total header will consist of a multiple of 4 bytes.

If the generic decap action is executed on a packet of type (1, 0x894f) it removes the entire NSH header from the packet, including any TLV context headers in case of MD type 2.

If the parameter new_pkt_type is set to (OFPHTN_ONF / OFPHTO_USE_NEXT_PROTO) the switch automatically derives the new packet type from Next Protocol field of the NSH header according to the following table:

Next Protocol	Packet Type	Comment

1	(1, 0x800)	IPv4
2	(1, 0x86dd)	IPv6
3	(0, 0)	Ethernet
4	(1, 0x894f)	NSH
5	(1, 0x8847)	MPLS

NSH packets with non-supported Next Protocol values will be dropped.

If new_pkt_type is set to any other value, it will override NSH's Next Protocol value. The outcome of that seems at best fragile. Should we support that at all?

If the decapsulated packet cannot be successfully parsed in accordance with its new packet type, the packet will be dropped as well.

Encap and Decap Actions For Ethernet

The encap and decap actions for Ethernet headers have already been described in chapter Generic Encap/Decap Use Case: Ethernet above.

Example OpenFlow Pipelines

In this example we have an SFC classifier and an SFF co-located on the same PTAP OVS bridge br-int. We assume the following connectivity to br-int:

Port 1	Ethernet port, ingress to classifier
Port 10	Ethernet port, egress to local SF
Port 11	Ethernet port, ingress from local SF
Port 20	Ethernet port, egress to gateway
Port 30	Ethernet port, egress local destination (10.10.20.20)
Port 100	VXLAN-GPE port (tun_dst = flow)

Suggested new ovs-ofctl command line syntax used in the examples:

- <Packet Type> = (Namespace, NS_ID)
  Short names: ethernet = (0,0), ipv4 = (1,0x800), ipv6 = (1,0x86dd), mpls = (1,0x8847), nsh = (1,0x894f), use_next_proto = (0,0xfffe)
- encap(<Packet Type>): Encap action with no extra encap properties
- encap(<Packet Type>,Prop1,Prop2...) where
  PropX is of the format {<PropClass>,<PropType>:<Val1>,<Val2>, …}
  Examples: {nsh,md_type:1}, {nsh,tlv_hdr: <Class>,<Type>,<Len>,<Value>}
- decap(): Decap action with new_packet_type = use_next_proto
- decap(<New Packet Type>)

Classifier flow entries

```
# Send Ethernet traffic from port 1 to the classifier table 10
add-flow br-int table=0,priority=10,in_port=1,packet_type=(0,0),vlan_vid=0 actions=goto_table:10

# Send TCP traffic to port 8080 to RSP with SPI 1000. Forward the packet to co-located SFF (table 30)
# Encapsulate the IP packet in NSH, not the Ethernet frame
add-flow br-int table=10,priority=10,tcp,nw_dst=10.10.0.0/16,tp_dst=8080
actions=decap(),encap(nsh,{nsh,md_type:1}),set_field:1000->nsh_spi,set_field:2222->nsh_c1,goto_table:30
# Send UDP traffic to RSP with SPI 2000, starting SI 25. Forward the packet to co-located SFF
add-flow br-int table=10,priority=10,udp,nw_dst=10.10.0.0/16
actions=decap(),encap(nsh,{nsh,md_type:2},{nsh,tlv_hdr:0xfff6,100,4,0x11223344}}),set_field:2000->nsh_spi,set_fi
eld:254->nsh_si,goto_table:30
```

SFF flow entries

```
# Ingress rules for traffic from local SF and remote SFF. Normalize NSH packets to L3
add-flow br-int table=0,priority=10,in_port=10,packet_type=(0,0) actions=decap(),goto_table:30
add-flow br-int table=0,priority=10,in_port=100,packet_type=(1,0x894f) actions=goto_table:30

# SFF forwarding rules
# Initial hop for SPI 1000 to locally connected SF via Ethernet
add-flow br-int table=30,priority=10,packet_type=(1,0x894f),nsh_spi=1000,nsh_si=255
actions=encap(Ethernet),set_field:11:22:33:44:55:66->dl_dst,output:10
# Second hop for SPI 1000 to remote SFF
add-flow br-int table=30,priority=10,packet_type=(1,0x894f),nsh_spi=1000,nsh_si=254
actions=set_field:999->tun_id,set_field:164.48.23.16->tun_dst,output:100
# Final hop for SPI 1000. Remove NSH and forward to final L3 destination
add-flow br-int table=30,priority=10,packet_type=(1,0x894f),nsh_spi=1000,nsh_si=253
actions=copy_field:nsh_c1->reg1,decap,goto_table:40
# First hop for SPI 2000, remote SFF
add-flow br-int table=30,priority=10,packet_type=(1,0x894f),nsh_spi=2000,nsh_si=254
actions=set_field:300->tun_id,set_field:8.8.8.8->tun_dst,output:100

# Final L3 forwarding at the end of SPI 1000
add-flow br-int table=40,priority=34,packet_type=(1,0x800),reg1=2222,nw_dst=10.10.10.0/24
actions=encap(Ethernet),set_field:11:22:33:44:55:66->dl_dst,set_field:66:55:44:33:22:11->dl_src,output:20
add-flow br-int table=40,priority=42,packet_type=(1,0x800),reg1=2222,nw_dst=10.10.20.20,
actions=encap(Ethernet),set_field:33:44:55:66:77:88->dl_dst,set_field:88:77:66:55:44:33->dl_src,output:30
```

Work Split

In this chapter we outline how to divide the entire feature into manageable work packages that can be developed and upstreamed independently and suggest a possible integration anatomy.

Work Packages

1 - L3 packets in kernel datapath (net-next)

- Versatile tunnel ports (GRE, any other?)

- Ethernet and L3 packets on Netlink API
- Push/pop_eth actions
- Some follow up needed: VLAN, MTU (Pravin and Jiri)
- Merge the net-next patches to OVS kernel module (Yi Yang)

2 - Layer 3 tunnel configuration

- OVSDB option on tunnel port (from Simon)
- Native tunnels in netdev datapath (from Simon)
- RTNETLINK api for kernel module in net-next and OVS (
    - Depends on RTNETLINK use in OVS (Jiri RedHat)
- Compatibility NETLINK api for OVS kernel module (Yi Yang)

3 - L3 ports in non-PTAP bridge

- Add packet_type fields to struct dp_packet and struct flow to replace base_header
- dpif-netdev
    - Adapt netdev providers to include packet_type in received packets
    - For native tunnel types GRE (and Geneve?):
        - Native tunnels in netdev datapath (from Simon)
        - Derive the packet_type from “Next protocol” field at tnl_pop
        - Set the “Next protocol” field from the packet_type at tnl_push
    - Map received packet_type to struct flow in miniflow_extract
    - Push/pop_eth datapath actions
- dpif-netlink
    - Derive packet_type from NL attributes Ethernet and Ethertype
- Handle non-Ethernet packets at boundary of ofproto-dpif:
    - Immediate push_eth for L3 packets from Layer3 tunnel ports
    - Drop Ethernet packets from Layer3 ports and vice versa
    - Pop_eth immediately before transmit to Layer3 port
    - Within ofproto-dpif continue to handle only Ethernet packet_type (0,0)

4 - PTAP bridge

- Ptap attribute for bridge in OVSDB and in ofproto
- Packet-type match field in ofproto (and ovs-ofctl), all OF versions
  Q: Can we use the same OXM in all OF versions? Or do we need NXM pre 1.5?
- Accept non-Ethernet packets from all ports transparently to ofproto-dpif-xlate
- Handle non-Ethernet packet_types in ofproto-dpif-xlate
  Supported types: IPv4, IPv6, MPLS. Should be easy to extend!
- Include packet_type OXM in Table Features Request/Reply
- Include packet_type in Packet_in
- Accept packet_type in Packet_out
- What about packets generated by OVS (BFD, LACP, CFM, …)?

5 - Basic EXT-382

- Generic encap/decap actions in ofproto and ovs-ofctl
- Specific encap/decap actions for Ethernet
  Invoking push/pop_eth datapath actions
- Basic framework for handling encap/decap properties in ofproto

6 - VXLAN-GPE tunnel

- Netdev datapath and kernel datapath (net-next and OVS)
- GPE option in VXLAN tunnel configuration
- Use different default dst port 4790
- Handle versatile payload using “Next protocol” field in RX and TX
- OVS kernel module may use compatibility Netlink API for GPE tunnel configuration

7 - NSH MD1 Match Fields

- Add packet type (1,0x894f) support in PTAP for NSH
- Add all NSH OXM match fields for MD1 to ofproto and ovs-ofctl
- Add NSH OXM fields to struct flow
- Parse NSH MD1 in extract_miniflow
- Match on NSH MD1 fields
- Set_field and copy_field/MOVE actions for NSH fields
- Optional: dec_nsh_si OpenFlow action

8 - Encap/Decap NSH MD1

- Specific encap/decap handling for NSH MD1
- NSH_MD_Type encap property
- NSH_OaM encap property
- push/pop_nsh actions in the kernel and netdev datapaths
- Clear NSH OXM fields at decap and trigger reparsing

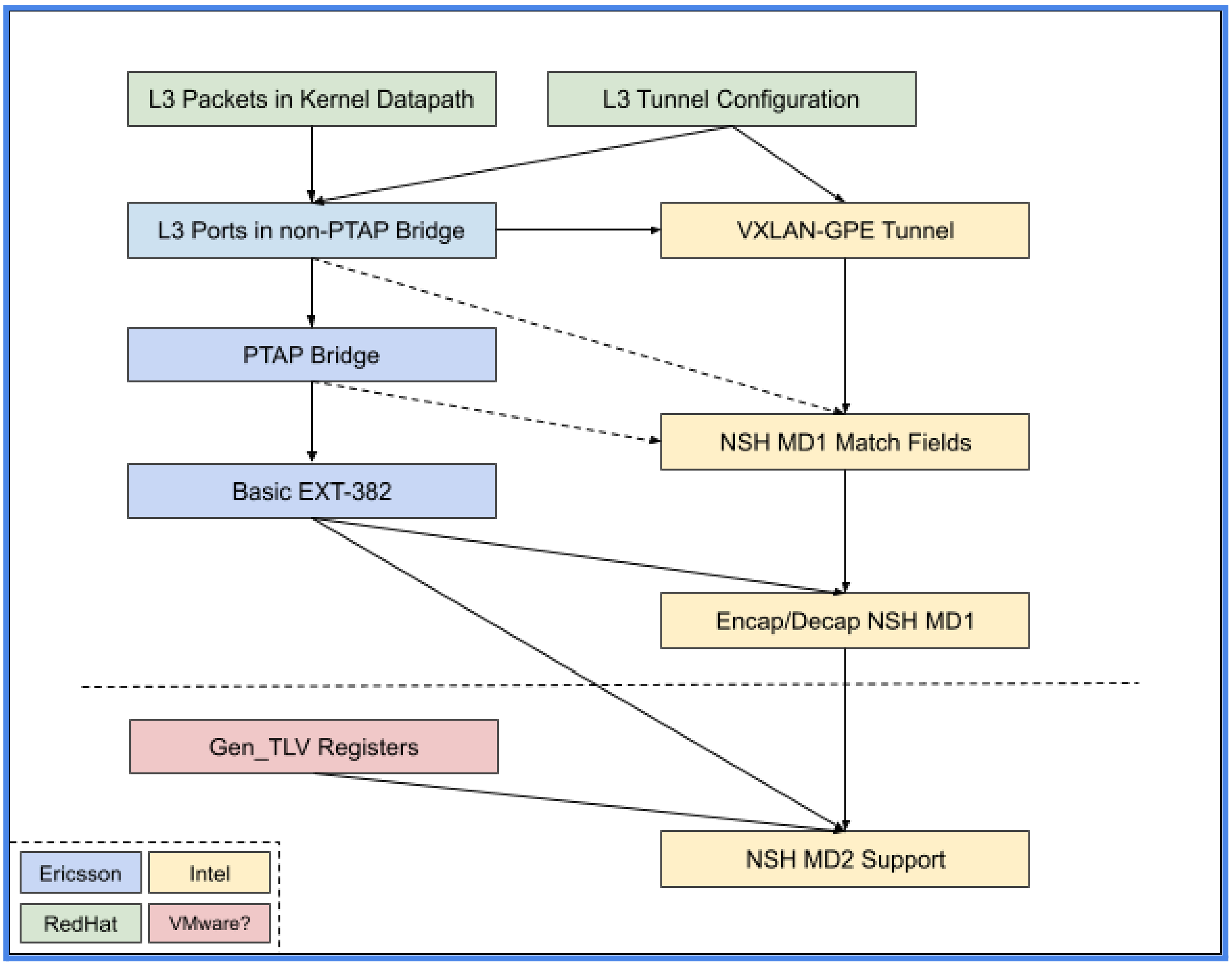
9 - GEN_TLV registers

- Separate set of TLV registers for packet header match fields
- New OF Add/Remove TLV Map command versions to configure mapping for protocols
- Datapath representation, ofproto-dpif representation and mapping between them
  Do not slow down processing of non-NSH or NSH MD1 packets!
- Match on gen_tlv registers
- Copy_fields/MOVE and set_field actions for gen_tlv registers

10 - NSH MD2 Support

- Parse NSH MD2 TLVs into mapped gen_tlv registers
- NSH_TLV_Context encap property to push MD TLV headers
- Enhance the push/pop_nsh actions in datapath to take a list of TLVs

Anatomy



The arrows show dependencies between the work packages and a possible integration order. The work packages above the dashed line comprise a self-contained solution for NSH with fixed size MD1 header. Support for MD2 can be added in a second phase.

The colors indicate a possible work split and are merely a first proposal.

Open Questions

- We would like to support PTAP, EXT-382 and NSH also as extensions to OF 1.3. Is that OK?
- Can we use the OXM classes and fields (to be) assigned for NSH also as fields in older OF versions (1.3, 1.4, 1.5) in OVS? Or do we need to assign Nicira extension fields for them?
  A: Use the standard code points

- Should we limit NS=1 packet types to known values (IPv4, IPv6, MPLS, NSH) or can we (in analogy to Ethernet) accept any NS=1 packet type and live with the fact that the OF pipeline can't match on anything except packet_type?
  A: Accept all valid Ethertypes (above 0x600) from tunnel ports that have the same protocol encoding as Ethernet.
- Can we assign an OXM class for GEN_TLV registers already even though they are not standardized yet and not sure if they ever will?
  A: Ben M-C to ask to reserve a class for this purpose. Possibility for standardization when stabilized.

Implementation Sketch

Impact of PTAP on netdev datapath

Struct pkt_metadata will be enhanced with a packet_type field comprising the 16-bit namespace and the 16 bit ns_type fields. Netdev providers must initialize that field with the correct packet type at rx.

Impact on Kernel datapath

The openvswitch kernel module uses the Netlink interface to pass packet together with its metadata between kernel and userspace. The kernel datapath with Jiri's patch encodes the packet_type implicitly in the OVS_KEY_ATTR_ETHERNET and OVS_KEY_ATTR_ETHERTYPE attributes. That way OVS userspace and kernel module can exchange information if it is about a L2 or L3 packet. This is sufficient for now and is necessary to be kept for backward compatibility. In the future, these attributes could be replaced by another attribute to indicate for instance L4 packets.

The datapath regards all tunnel ports as versatile. Datapath rules need exact match against L2 or L3 property. Packets that need specific handling will be sent to userspace via slow path.

Impact of PTAP on ofproto-dpif

The struct flow will be enhanced with a packet_type field comprising the 16-bit namespace and the 16 bit ns_type fields.

In a non-PTAP bridge the packet_type will be implicitly set to (0, 0) Ethernet.

Prerequisites on packet_type (0, 0) Ethernet will be added for Ethernet OXM fields (OF_ETH_DST, OF_ETH_SRC and OF_ETH_TYPE). Same for VLAN OXM fields.

Alternative pre-requisite packet_type=(1, 0x800) will be allowed for IPv4 match fields.
Alternative pre-requisite packet_type=(1, 0x86dd) will be allowed for IPv6 match fields.
Alternative pre-requisite packet_type=(1, 0x8847) will be allowed for MPLS match fields.

Impact of PTAP on ofproto

TODO: OF protocol impacts for packet_type match field in OF 1.3

Configuration of tunnel port mode

Add a new tunnel port option “layer3”. This boolean option could be used only on ports of non-PTAP bridges. The default value would be “false” indicating L2 tunnel port due to backward compatibility.
For PTAP bridges, there is no need to introduce such tunnel port option since all ports are versatile.

EXT-382: Generic Tunnel Encap/Decap

By providing general encap/decap OF actions a controller will be able to modify the packet_ type of a packet in the OF pipeline by adding an outer header for encapsulation or by removing an outer header from the packet and exposing the inner packet to the pipeline.

The following specific headers shall be supported in the generic encap/decap actions:

- Ethernet
- NSH (metadata formats MD1 and MD2)

Structuring into patch sets

TO BE DONE:

1. User-space patch to allow configuration of L3 port and stitch the L3 tunnel port to Ethernet-only (non PTAP) bridge. This involves pushing a dummy Ethernet header at rx from L3 tunnel port and popping L2 header (incl VLAN tags) at TX to L3 tunnel port.

Open Questions

Q: Is there a way in OF to signal the PTAP property of a bridge.
A: According to OF 1.5.1 packet_type can be configured per table not per bridge. The OFPMP_TABLE_FEAURE request can be used to query/configure the list of supported packet types. In order to do a query, the OFPMP_TABLE_FEATURE request should be sent with empty body. In order to configure the tables, the OFPMP_TABLE_FEATURE request should be sent with OFPTFPT_PACKET_TYPES property for each table to be configured. This property contains the set of Packet Types Match Field values to be supported by the table.

Miscellaneous Notes

[Jan]: the flow field next_base_layer is only applicable in user-space datapath. It is populated when a GRE packet is parsed (extract_miniflow()) to indicate whether the carried payload is L2 or L3. This is used in the user-space tunnel lookup (tnl_ports.c) to select the right GRE tunnel port if there is one for L2 and another for L3 payload. It is not used elsewhere.

1. Kernel datapath tunnel configuration patch (Cascardo) needed (RTNETLINK API)
2. User-space patch for configuration of l3 tunnel flag (dependency on Cascardo patch set)
3. User-space patch to stitch L3 tunnel to L2 pipeline (push/pop at tunnel boundary)
4.

References

[OpenFlow 1.5.1]

OpenFlow 1.5.1 Specification

https://www.opennetworking.org/images//openflow-switch-v1.5.1.pdf

[EXT-112] Packet Type-aware Pipeline

https://rs.opennetworking.org/bugs/browse/EXT-112

[EXT-382] Generic tunnel Encap and Decap

https://rs.opennetworking.org/bugs/browse/EXT-382

[Layer3 Tunneling kernel patch]

[PATCH net-next v12 0/9] openvswitch: support for layer 3 encapsulated packets

http://mail.openvswitch.org/pipermail/ovs-dev/2016-October/080697.html

[Layer3 Tunneling user-space patch]

[PATCH v12 rebased 0/3] userspace: Support for layer 3 encapsulated packets

http://mail.openvswitch.org/pipermail/ovs-dev/2016-October/080828.html

[Johnson Li NSH Patch]

[RFC PATCH v2 00/13] Add Network Service Header Support

http://mail.openvswitch.org/pipermail/ovs-dev/2016-July/074922.html

[PTAP prototype]

Jean's post with link to his PTAP prototype

http://mail.openvswitch.org/pipermail/ovs-dev/2016-February/066842.html

[ONF Registry]

https://rs.opennetworking.org/wiki/display/PUBLIC/ONF+Registry