Proc. of the Intl. Conf. on Future Trends in Computing and Communication -- FTCC 2013

ISBN: 978-981-07-7021-1

Describing MPLS in an Open Flow enabled Software Defined Networks

[Sandeep Singh, R.A. Khan]

Abstract

The main advantage for taking this proposed approach is that it is a protocol and application agnostic. This will not only work for MPLS, but also for other tunneling protocols such as GRE, or L2TP. It will also work for L2 protocols such as PBB, and applications such as VPLS. Variants such as Nested LSPs are handled as well in the same manner. Our goal is to make the switch as protocol/application unaware as possible. Instead the protocol and applications reside in the controller only. Our Specification is Implementation independent (aka Platform Independent), as a specification should be. At the same time, we do leverage our extensive experience with implementations to make sure that our specification is not devised in vacuum. We have gone to pains to make sure that the implementations will have the flexibility to address their own specific constraints and optimizations while working within the boundaries of the specification.

Keywords: MPLS, Open Flow Switch, SDN**.**

Sandeep Singh, Research Scholar
Department of Information Technology
Babasaheb Bhimrao Ambedkar University, Lucknow.
INDIA

Dr. R. A. Khan, Associate Professor
Department of Information Technology
Babasaheb Bhimrao Ambedkar University, Lucknow.
INDIA

I. INTRODUCTION

The current OF spec (v1) only defines the Flow Table. The outgoing ports are represented by a bit map of ports. This will not be enough to handle certain functionality that is in common use. For example IP Multicasting will require that a packet be replicated to multiple outgoing VLANs. OFv1 cannot handle that, because the FT can specify only one outgoing VLAN. Other examples that will not work include VPLS, Point-to-Multipoint Tunnels etc [1].

Today, all packet processing in the OF architecture is specified in the FT. There is a "mention" of a Virtual Port, but it is NOT defined anywhere. It is clear that the term Virtual Port is being used as a Place Holder. It is used in an "abstract" way to explain functionality that in reality is not possible or even exists in OF today. At Ericsson Research we are in the process of defining and implementing MPLS capability for OF. Therefore we don't have the luxury of treating the Virtual Port as a vague notion that will somehow provide the functionality that is required by tunneling (such as Nested Tunnels, VPLS, FRR etc).

The Virtual Port Table (VPT), as described earlier, extends the architecture in a manner that is very similar to the Flow Table. One can think of the VPT as the FT for packets as they egress the box. It is a symmetrical manifestation of the FT. Instead of matching on various fields of the packet, a VPT is indexed by the VP handle. This makes it easier to implement and cheaper. It can be thought of as the match on the VP index. The VP entry yields a set of table actions very much like the FT entry does. These actions are necessary to put the packet on the port. One important capability of the VPT is the optional chaining of VP entries. This allows for Hierarchical Ports. For example a PW port will be chained to its parent Tunnel port, which is chained to its parent

Proc. of the Intl. Conf. on Future Trends in Computing and Communication -- FTCC 2013

ISBN: 978-981-07-7021-1

physical port. Port chaining is a powerful concept that can handle any complexity of encapsulations. But it comes at a price. It is essentially a linked list and that means that its processing takes O(n). This is not acceptable for the typical cases of encapsulation. The forwarding path processing is sacred for the typical cases since it impacts the Line Rate. Therefore we allow that a VP entry can short cut the chain. It does so, by compressing the actions of all the chain into one entry. For example a PW port can push the VPN and Tunnel labels from one entry to avoid chaining. Another example would be where a Tunnel over a Link Bundle, keeps the cached active or hashed constituent of the bundle in the Tunnel entry [2].

II. BACKGROUND

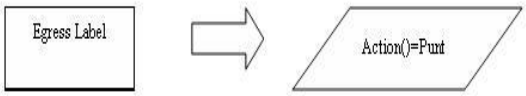
A. *Outgoing Port List:*

The Port Table also contains the block of Outgoing Port Lists (OPLs). The OPL is used for replication, which is required in a number of applications such as Multicast, VPLS etc. Each replication will represented by an entry, and all the entries of a particular OPL will be contiguous. The last replication entry in the OPL will have EOL (End Of List) flag.

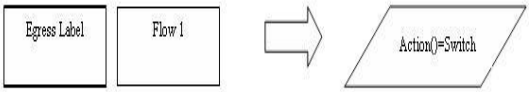
B. *On Demand Virtual Ports:*

Typically a VP will be setup preemptively, either by configuration, or signaling. However the spec does allow the creation of VPs on the fly, or on demand.

C. *Initial tunnel egress entry (Tunnel VP):*



The above Flow Table entry embodies the Tunnel VP. Now if a Subscriber flow egresses the tunnel it will get punted to the controller, which will install the specific entry for it:



The more specific entry above is also a VP, in a hierarchy of ports. In this case, the new on demand VP is a Subscriber VP. The ability to increase the match specificity allows us to de-capsulate and forward in a single lookup cycle. As presented so far Flow Table and Port Table have identical actions. Some people have pushed back that certain new actions required for encapsulation/decapsulation (such as push/pop) should not be added to the Flow Table. This is understandably so since this is motivated by a desire to keep the new functionality quarantined in the new entity described as the Port Table. The case has also been made that we can defer the action to the port table, so why add it to the flow table. The author strongly disagrees with the above argument. There is no technical reason for quarantining the new actions required for tunneling to the port table. On the other hand common functionality in both tables helps with a number of cases. For example, in the case of VPNs, an incoming packet can immediately be tagged with its VPN label in the flow table, thereby establishing the context for the packet before it gets injected into the switch fabric. In this manner the outgoing virtual port is the common transport tunnel (transporting all PWs) as opposed to a PW. Since a box can have 10s of thousands of VPNs, this can lead to a big saving in the Port Table size and complexity. It also simplifies the management of entries by the controller. By providing symmetry between the Flow Table and Port Table, the architecture provides the necessary flexibility to the switch implementer, without introducing unique functionality for the two tables.

D. *Ingress and Egress Flow Tables:*

Finally the author is going to make a very strong case that the Port Table should really be the same as the Flow Table. The port table that has been described so far

Proc. of the Intl. Conf. on Future Trends in Computing and Communication -- FTCC 2013

ISBN: 978-981-07-7021-1

is really an Economical Flow Table, since the match is only on the outgoing port. While this works for the use cases presented so far, there are many important functions that require that the Port Table should be the same as the Flow Table. Examples include Outbound ACLs, Lawful Intercept, Multi-Stage Forwarding etc. Lawful intercept is an example of functionality where the Government requires that it be supported for sale to carriers. Port Chaining will still be required for Port Hierarchy, Multicast etc. The spec should provider the flexibility for an implementation to choose the level of complexity in the Port Table depending upon the market niche it is addressing. The range would be from:

- No Port Table (support only L2 and unicast L3/L4 functionality)
- Economical Port Table (adds tunneling and L3 multicast)
- Egress Flow Table (adds Outbound ACLs, LI, Multi-Stage Forwarding etc.)

III. MPLS OPERATIONS

An MPLS path have three types of nodes:

- **Ingress**: tunnel encapsulation
- **Transit**: label switching
- **Egress**: tunnel decapsulation

The actions required at these nodes are:

- **Push**: path ingress
- **Swap**: transit
- **Pop**: path egress
- **PHP** (Penultimate Hop Popping): used to avoid Pop at the egress node
- **Swap & Push**c
- **Pop & Swap**: nested path egress

Further functionality required at the nodes:

A. Forward to the MPLS next hop

: After performing the action on the labeled packet, it is forwarded to the next node in the MPLS path. This is done by re-writing the destination MAC to the MAC address of the next hop. Also the source MAC of the packet is changed to the source MAC of outgoing MPLS interface.

B. TTL handling:

TTL handling requires a number of operations: 1) Move the TTL from the IP header to the MPLS label at ingress node, 2) Decrement the label TTL at the transit and egress nodes, 3) Move the TTL from the label to the IP header at the egress node, 4) If the incoming label TTL is 1 at the transit or egress node then do not forward the packet. Instead punt it to the controller for further handling [3].

IV. MPLS ACTION

Once an MPLS flow has been matched, Table Actions (TA) must be implemented, which will provide the necessary MPLS functionality. The MPLS label will need to be imposed, disposed or swapped. These actions can be considered as a case of generalized header re-write. The simplest case would be that of a single labeled packet that requires a swap. In that case, the swap would be nothing but a label re-write. It does get trickier for the other cases. For example, Push would require a label to be added. The header is split into four levels, with a buffer associated with each level:

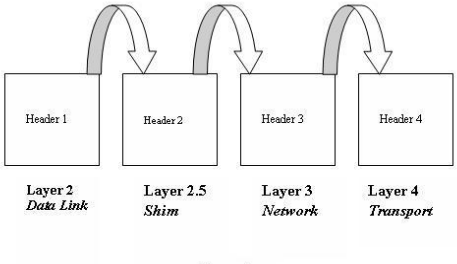


Figure 1

This arrangement makes it easy to insert/remove and manipulate header fields. For example, a TCP flow coming into the MPLS cloud on an Ethernet port will be seen as:

Proc. of the Intl. Conf. on Future Trends in Computing and Communication -- FTCC 2013

ISBN: 978-981-07-7021-1

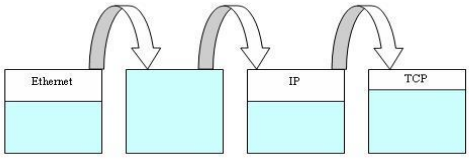


Figure 2

Now tunnel encapsulation is done by simply adding the MPLS label in the level 2 (Layer 2.5) buffer:

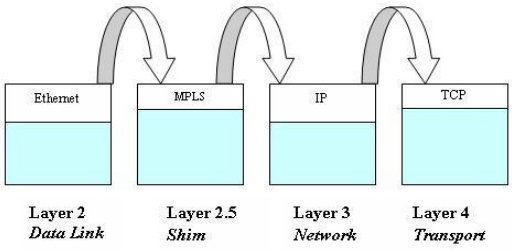


Figure 3

At the transit nodes the Shim buffer gets re-written. Finally at the Egress node the tunnel encapsulation is removed by making the Shim buffer null (as in figure 1). This generalized mechanism can be employed for all tunnel encapsulations (GRE, IPinIP, L2TP, MPLS Label Stacks, QinQ, PBB etc).

v. POP aka Tunnel Egress:

The logical action for POP, which is a tunnel egress operation, is to remove the tunnel encapsulation and then do a second lookup. This can be modeled in OpenFlow by installing the tunnel egress matching flow with an action of punting to the controller. For MPLS that would be the label advertised by the egress node. The first packet for a flow being transported by the tunnel will be punted to the controller. The controller will then install a more specific flow entry that will include the inner header along with the switching action.

- Actions:
- Pop (buffer shim, 4 bytes)
- Rewrite IP TTL from MPLS TTL
- Outgoing Port = As desired for the FEC

CONCLUSION

Our design philosophy is having a general scheme which is independent of a given protocol and a set of actions associated with the virtual port that are within the scope of a lightweight Open Flow switch. It is also found that this philosophy is having a great flexibility, and scalability to handle future needs. The virtual port (VP) falls into the general context of ports on a switch. An Open Flow switch will maintain a Port Table. Each port will be identified with a handle that can be used to index the Port Table (PT). Each entry on the port will contain the set of actions required to put a packet out on the port, a pointer to port data blocks and parent port (to support a hierarchy of ports).

References

[1] Saurav Das, Unified Control Architecture for Packet and Circuit Network Convergence, PhD Thesis, Stanford University, June 2012.

[2] Saurav Das, Ali Reza Sharafat, Guru Parulkar, Nick McKeown, MPLS with a Simple OPEN Control Plane, invited talk at Packet Switching Symposium at OFC/NFOEC'11, Los Angeles, March 2011.

[3] Ali Reza Sharafat, Saurav Das, Guru Parulkar, Nick McKeown, MPLS-TE and MPLS VPNs with OpenFlow, demonstration at SIGCOMM, Toronto, August 2011.