Internet Research Task Force
Internet Draft
Intended status: Informational
Expires: February 03, 2014

P. Ashwood-Smith
Huawei
M. Soliman
Carleton University
T. Wan
Huawei

July 03, 2013

SDN State Reduction

draft-ashwood-sdnrg-state-reduction-00.txt

Abstract

This document makes the argument that to support the centralized control of a substantial number of forwarding devices (as Software Defined Networking (SDN) proposes) that the scale, speed, cost and general quality of such a solution will be improved by reducing the state needed to be distributed into the network of devices by the controller(s). To this end we re-visit forms of Source Routing (SR), in particular Strict Link Source Routing (SLSR) and suggest that light weight SLSR could allow substantial reduction in controller burden while potentially reducing the costs/complexity on forwarding devices. We discuss some simulation results that demonstrate these advantages and how the advantages grow substantially as the network diameter grows. We also look at various implementation possibilities including existing IPV4, V6, MPLS, new/modified MPLS vs. something brand new that could possibly be implemented with new SDN technology like Protocol Oblivious Forwarding-POF.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on December 3, 2012.

Table of Contents

1. Terminology

ATM	Asynchronous Transfer Mode (a cell based network)
BGP	Boarder Gateway Protocol
CSPF	Constrained Shortest Path First
DOS	Denial of Service (attack)
ECMP	Equal Cost Multi Path
flow	Logically related packets following the same path
IS-IS	Intermediate System to Intermediate System
LACP	Link Aggregation Control Protocol
LAG	Link Aggregation
Loose	A source route that enumerates only some of all hops
MPLS	Multi Protocol Label Switching
MPLS-TE	MPLS Traffic Engineering.
NPU	Network Processor Unit (programmable forwarding)
OpenFlow	Open data path programming protocol
OSPF	Open Shortest Path First
PCE	Path Computation Element (used with MPLS-TE)
PNNI	Private Network to Network Interface (link state ATM)
POF	Protocol Oblivious Forwarding - more generic OpenFlow)
RSVP-TE	Resource Reservation Protocol - Traffic Engineering
SDN	Software Defined Networking (as per [OPENFLOW])
SDN-domain	Set of forwarding devices controlled as a unit.
SR	Source Routing - enumerating hops to traverse
SLSR	Strict Link Source Routing - enumerating links [SLSR]
SPF	Shortest Path First - (Dijkstra etc.)
SSRR	Strict Source Record Route - IPV4 header option 9
Strict	Source route that enumerates every hop(unlike Loose)
TE	Traffic Engineering
VFI	Virtual Forwarding Instance (layer 2)
VPLS	Virtual Private Lan Service
VRF	Virtual Routing and Forwarding (layer 3)

2. Introduction

The centralized control of a network is not a new idea. Indeed centralized control was widely deployed in voice networks and some early data networks but of course gave way to distributed control for IP.

Centralized computation is however still widely used for traffic engineered networks, like MPLS-TE and GMPLS where a Path Computation Engine (PCE) makes use of a global view of a sub-network and its resource usage for the planning of new paths and their resources. The data path state distribution with these models is however not initiated centrally and relies on protocols like RSVP-TE to install the hop by hop state. In fact this form

of distributed control with centralized traffic engineering computations is the norm today.

Notwithstanding the massive deployment of this kind of hybrid distributed/central control, we have in the last several years seen a huge resurgence of interest in fully centralized control of at least a set of forwarding devices [ONF] [OPENFLOW] with Software Defined Networking (SDN). This SDN proposes a central controller (or controllers) using IP protocols such as TCP to talk to a set of arbitrarily interconnected (and cheap/dumb) forwarding devices (SDN-domain) and which is responsible for the configuration of the majority of forwarding state on those devices. This state may be produced either as a result of pro-active configuration, or based on re-active responses to packet flow indications from the forwarding devices themselves.

Since this central controller has knowledge of the entire sub-network of devices, and potentially of the traffic demands into/out of the sub-network, it can perform a variety of path optimization computations similar to CSPF/MPLS-TE/PCE/GMPLS, or even more elaborate forms of optimization (trading flows against each other rather than individually optimizing them, exploiting quiet areas of the network to offload busy areas etc), the output of which is forwarding state for all meta flows in the entire sub network of devices and a sub network which more optimally meets the desired local constraints. One such deployment reports a substantial increase in network utilization from 30% to 70%-90% [SDNGOOG].

A central controller can also more effectively solve problems such as bin-packing and path blocking [SDNGOOG], which occur when flows are optimized individually with greedy type algorithms rather than considering other orderings of the flows. The finer grained ability to place traffic can also permit much more detailed placement of traffic after a failure, including traffic not directly affected by the failure but the replacement of which is critical to achieving fair/efficient use of the remaining bandwidth subsequent to the failure.

Since the output of the controller is much closer to a TE (Traffic Engineered) type solution from a PCE (Path Control Element) than an SPF (Shortest Path First) solution the controller cannot simply install destination based forwarding entries. A controller either needs to install tunnels that follow the explicit routes it wishes and then map traffic to those tunnels at the edges, or it must install n-tuple < <source IP> <destination IP> <source port> <destination port> etc.> state and configure these n-tuple matches on every hop along the desired path. Packets which fail to match

an n-tuple are either discarded or sent to the controller.

In the normal case of SDN (as given in [OPENFLOW]) the controller is required to send configuration information to all devices along the path from ingress of the SDN-domain of this controller to the egress of that SDN-domain, alternatively a tunnel setup protocol like RSVP-TE is required to be triggered to distribute the per hop state between the ingress and egress.

This draft proposes that since the controller knows the exact end-to-end path (down to the level of the links it wishes the packets to traverse) and that the diameter of an SDN-domain is likely to be a reasonable number of hops, that the controller should instead simply insert into a header the exact links it wishes the packet to traverse and thereby not have to deal either with per hop n-tuple state installation (very expensive) or with MPLS tunnel installation via RSVP-TE(complex). Such a mechanism also eliminates any concerns about Equal Cost Multi Path (ECMP) and/or Link Aggregation (LAG) as the controller can place traffic on exact links.

Operations, Administration and Management (OAM) is also greatly simplified since data packets will flow on invariant paths that are known by both ends of the flow and can be the same as any OAM packets that probe the flow. This OAM "fate sharing" property is widely valued by network operators and considerable effort has already been expended to permit similar fate sharing between OAM and data paths with other carrier scale networking protocols such as 802.1ag and MPLS-TP. Of course if a controller does not wish to enforce symmetry and congruence it need not.

3. Logical Example

The following is an example of an idealized strict link based source routing (SLSR) forwarding. We talk about possible implementations including MPLS methods after looking at the logical ideal.

Consider the simple 7 node network shown in Figure 1 below. Here the nodes are named {A, B, C, D, E, F, G} and where each node has locally numbered interfaces named {1, 2, 3, 4, 5, 6}.

For example node A has interfaces named 1, 2, 3, 4 and where interfaces 4 and 2 both go to node B. Node B has local interfaces 1, 2, 3, 4 and 5 but the two interfaces going back to node A are locally named 1 and 3. Clearly node interface names are likely (but not necessarily) different at both ends of a link.

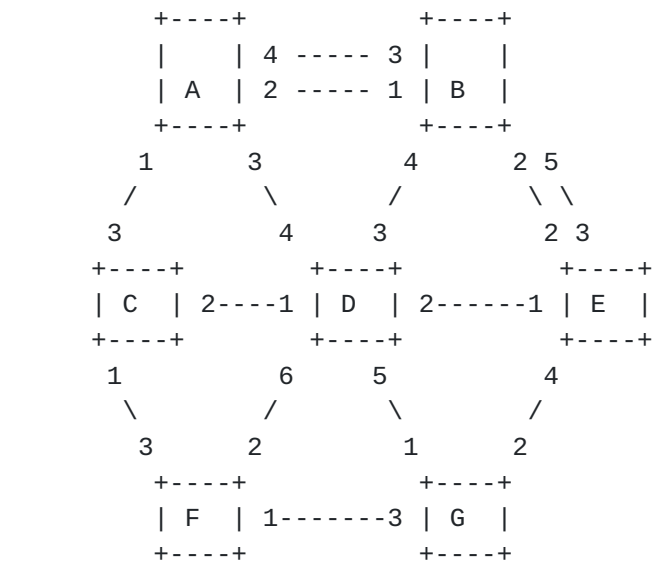


Figure 1 - simple 7 node network with local link identifiers.

4. Expressing a Path

A path through a network labeled as per Figure 1 can clearly be expressed as a sequence of link names (an SLSR).

For example, between nodes C and E the following are all valid paths.

```
C.3 -> A.2 -> B.2
C.2 -> D.2
C.1 -> F.2 -> D.2
C.3 -> A.4 -> B.5
```

Now since the links lead unambiguously to a known node, the paths can be more compactly expressed without the node names as follows:

```
{3,2,2}
{2,2}
{1,2,2}
{3,4,5}
```

As long as we know the origin of the path (in this case node C), the list of link names unambiguously identifies a path and an egress point. In addition it identifies unambiguously which link from among parallel links between neighbors should be traversed. Of course it is possible to give a name to the set of links that all attach to the same neighbor and thereby leave the exact link in that path deliberately ambiguous and thereby subject to a local forwarding decision as to exactly which link in the set to follow.

Each path of course also has exactly one perfectly symmetric reverse. Note that the symmetric reverse path is not simply the same list of link names in reverse order. A reverse path has to be specified from the opposite end of the path so in this example the origin has to be E.

The forward and corresponding reverse paths are therefore.

C->E	E->C
{3,2,2}	{2,1,1}
{2,2}	{1,1}
{1,2,2}	{1,6,3}
{3,4,5}	{3,3,1}

Various very efficient encodings of these kinds of paths in source routed headers are possible. Even a simple encoding using 8 bits per hop can encode every path in a large 8 hop network with fewer bits than an IP in IP tunnel.

5. Computing a Path

It should be obvious that the output of any graph based computation which has as its goal various optimization criteria for flows can express its results as a series of such paths where each path is expressed as a Strict Link based Source Route (SLSR). This includes multiple different metric Dijkstra computations (i.e. shortest path, multi topology shortest path), CSPF type and of course more elaborate linear-programming or other convex type optimizations.

The expression of the path as an SLSR imposes no constraints on the type of computation being performed except possibly in path length. However in any real network under the control of a single controller it is not likely that path length would be a real issue unless perhaps unreasonably large link names are encoded.

Convex and linear-programming type solutions to traffic placement are of particular interest because to do a good job they must exploit a considerable number of paths through a network (many more than shortest). These algorithms take the matrix of ingress/egress flows in a network together with all the usable paths between all sources and destinations and will assign percentages of the ingress/egress flows to the available paths in ratios that can optimize a number of simultaneous constraints. For example they can optimize the network's total throughput, the average link utilization, the fairness of the bandwidth available to each flow and can even optimize different linear and non linear combinations of those goals. What is interesting about all of

these kinds of optimizations however is that they need access to all of the reasonable paths across the network since it is by making trade-offs between busy and less busy parts of the network that they achieve their goals. Unfortunately the number of paths (shortest or otherwise) in a network grows exponentially with network size and with it the state distribution problem (or burden) the controller must deal with.

It is important to make the distinction between a flow and a path. This draft concerns itself not with the immense numbers of micro flows but with the very large numbers of paths required to be supported onto which those micro flows are then aggregated. A set of micro flows can be treated as a single flow, and a single flow has a unique path through the network.

6. Downloading Forwarding State

A controller likely takes as input the fields that identify the flow and its various statistical attributes. The controller then likely computes an end to end path for this flow either based on the single flow's attributes (in a re-active manner), or on more global knowledge of multiple flow attributes (in a pro-active manner). Flows may be meta (many micro flows) or individual micro flows depending on the implementation and its scale. The output of course is just a list of links that must be traversed for this flow together with matching rules to identify the flow at the ingress.

The controller then delivers the flow matching rules and the Strict Link Based Source Route to the *single* node where the flow is to be encapsulated (i.e. where the flow first enters the SDN-domain).

The fact of only having to communicate with the *single* node at the head end of the path means that the controller experiences a reduction in its work load directly proportional to the number of hops in the path (as compared to traditional SDN which must program every hop along the path).

Intuitively this translates to the following I/O burden reduction at the controller based on the number of links that must be traversed per average path.

#Avg Path Len	% I/O Burden Reduction
1	0%
2	50%
3	66%
4	75%
5	80%
..	..
N	(100-100/n) %

Since forwarding state download is typically a substantial part of a "normal" routers' re-convergence time, it seems reasonable that this will become a similar bottleneck for a central controller and quite possibly be further aggravated by the increased delays and larger amount of state that the central device must deal with.

As a result this reduction in state and I/O burden should have a marked impact on convergence times assuming there are appropriate forwarding mechanisms that can implement the Strict Link Source Route (SLSR). Note also that the position of the controller relative to the ingress/egress nodes is now more important than its position relative to all nodes. Therefore studies as to controller optimum placement as defined by the Controller Placement Problem [PLACEMENT] would require different optimization goals.

An additional 50% reduction can also be obtained should the implementation of the forwarding be able to reverse the path on the fly. Such a reversal permits the implicit communication of the desired reverse path to the receiver thereby eliminating communication with the controller to obtain a reverse path. Of course if symmetry is not desired this further optimization is not possible.

For example, consider a network with 1000 nodes. It therefore has O(1,000,000) meta flows and assuming 10 possible paths for each flow has O(10,000,000) ingress forwarding entries that must be centrally configured (its burden). If each path on average takes 5 hops then the burden on the controller grows 5 fold to O(50,000,000) entries but with SLSR the burden remains at O(10,000,000). If path reversal is supported and symmetric routing is desired then the burden with SLSR drops further to O(5,000,000).

Simulations done by one of us in [SRSDN] provide additional weight to the above arguments. In particular we simulated for various network sizes and diameters the differences between hop by hop SDN and SLSR and saw up to 3 x performance improvements in convergence times with SLSR. There were also a number of other benefits such as a markedly reduced standard deviation in convergence times for the different nodes (81% decreases) and a significantly reduced sensitivity to the placement of the controller (80% reduction in standard deviation). The performance improvements can perhaps better be understood by an analogy comparing the work required to fill in the area of an object (traditional SDN) vs. simply drawing the circumference of that object (SLSR). Since the circumference varies as a function of the diameter but the area varies as a function of the diameter squared the relative burden reduction with just dealing with the circumference (the edge of the network) becomes apparent. In fact in this simulation study we varied the radius and then plotted the relative convergence times of SLSR and traditional hop by hop forwarded SDN and saw a ratio of convergence times as a function of radius that indicated a trend towards 1/R as expected. Simply stated, the bigger the centrally controlled network the better source routing performs compared to hop by hop.

7. Logically Forwarding SLSR

There are three distinct phases to be performed to logically forward unicast SLSR. These are similar to any tunnel technology and consist of 1) Ingress Encapsulation, 2) Tandem Forwarding, and 3) Egress Decapsulation and Forwarding. We address the generic concepts first before looking at possible existing or new encapsulations and their applicability.

Multicast SLSR is also possible (but with limitations to keep the header sizes from growing too large) and is briefly discussed after unicast.

7.1. Ingress Logical Unicast Forwarding

Here the flow information, likely IP header(s) + UDP/TCP header(s) is looked up and a sequence of link identifiers and a current hop must be placed on the packet, the packet must then be forwarded to the first of those links. This operation is identical to almost every tunnel protocol except that IP ECMP and/or LAG hash would potentially be unnecessary because the first link name would often resolve to a physical link not a LAG bundle. For example:

```
+----------+----------+---------+--------+--------+
|SrcIP     |  DstIP   | SrcPrt  | DstPrt |   SLSR |
+----------+----------+---------+--------+--------+
|192.0.2.4 |192.0.2.9 |  1000   |  98    |{3,2,5} |
|192.0.2.4 |192.0.2.9 |  1001   |  99    |{3,4,5} |
+----------+----------+---------+--------+--------+
```

Of course nothing precludes the use of LAG and the link identifier therefore identifying an entire LAG bundle rather than an element of that LAG. In fact it is possible to simultaneously support both concepts so that some traffic can be forwarded to the entire LAG while other traffic could be placed on a particular LAG bundle member at the discretion of the central computation.

7.2. Tandem Logical Unicast Forwarding

At tandem devices the operation would start by incrementing the current hop in the packet header (shown with a ^ symbol) and then forward to the link identified in the new current hop. If we support reversal, we change the previous link name to the local link name for that link. For example, referring to Figure 1 (and disregarding non relevant headers/options) after matching the first flow tuple at the ingress node C the packet is encapsulated with the SLSR header {3,2,5} and then leaves node C on interface 3 toward A. Then:

Packet arrives at node A on local interface 1 where it looks like this:

```
+-------------------------------------~~~------------+
| 3 | 2 | 5 | 192.0.2.9 | 192.0.2.4 |..   <payload>   |
+-^-----------------------------------~~~------------+
```

Current hop is incremented while previous hop is changed to local interface name (3 changes to a 1).

```
+-------------------------------------~~~------------+
| 1 | 2 | 5 | 192.0.2.9 | 192.0.2.4 |..   <payload>   |
+-----^---------------------------------~~~------------+
```

Packet is forwarded to interface for current hop i.e. 2.

Packet arrives at node B on local interface 1.

Current hop is incremented while previous hop is changed to local interface name 1 (2 changes to a 1).

Packet is forwarded to interface for current hop i.e. 5.

Packet arrives at node E on local interface 3.

Current hop is incremented while previous hop is changed to local interface name (5 changes to 3).

We are at the end of the path, so egress processing begins.

One additional step not described above is a reverse path check. Prior to substituting the reverse link identifier into the SLSR header, the link identifier from the neighbor can be validated and the packet discarded if the neighbor link identifier in the packet is incorrect for the port the packet arrived on. This would reduce the chances of mis-delivery of the packet should a link identifier change or a link destination change while a packet is in flight.

7.3. Egress Logical Unicast Forwarding

Here the operation consists of normal IP/Ethernet etc. forwarding based on the IP destination / MAC or other ACL rules. Basically the SLSR header is stripped and the packet is submitted to the Virtual Forwarding Instance, or Virtual Forwarding Function (VFI or VRF) for further processing.

Optionally the link identifier from the neighbor can be validated against what is expected and the packet discarded in the case of a

mismatch. This reduces the chance of mis-delivery as in the tandem case.

In addition, if path reversal is supported, the reverse path is compared against the current reverse path for this reverse flow and if it has changed the local forwarding state for the reverse flow would be updated. This would allow the head end to always dictate the forward and reverse path to be used for all packets in the flow without involving the controller on the egress side (and of course not needing to communicate with any tandem device).

Processing the reverse flow/path in this manner means that a flow is already present for the reverse direction without having to re-actively or pro-actively consult the controller. This results in a further 50% reduction in controller load. In the case of asymmetry this optimization is of course not possible.

8. Logical Multicast Forwarding SLSR Packets

Multicasting packets usually involve one of two approaches.

The first approach simply re-uses unicast and sends multiple copies to a pre-determined list of receivers. There is little to discuss with this approach as we can replicate SLSR based unicast packets just as easily as any other tunneling mechanism. Clearly such a serial unicast approach has nearly identical bandwidth overhead as other protocols like VPLS which also use this serial unicast mechanism.

It is therefore interesting to look at more efficient methods that involve the second multicast mechanism, which uses replication points in the network. These replication points are chosen so that copies are more efficiently made thereby eliminating multiple copies of the packet traversing any given link. Various logical tree structures are usually involved e.g. STP, SPB, TRILL, PIM, MOSPF etc.

These tree based mechanisms could in theory be implemented without requiring tandem state as an SLSR by introducing a branch point concept into the list of indexes. In this manner a complete tree as a pre-order traversal could be encoded along with the packet payload. It is not difficult to define a variety of different encodings that would accomplish this. The obvious objection to such a scheme is the sheer size of header required especially where a large network with many multicast receivers is concerned. It is therefore unlikely to be practical to encode any large tree of receivers and the SLSRs between them in any single header.

This leads to a hybrid approach which would encode a subset of the tree, say a single replication point and 5 or so recipients. This little tree or 'tree-let', would efficiently get a single packet to 5 (or some suitably small number) of recipients with an SLSR to the replication point and then SLSRs to each of the receivers. Such an encoding is much more reasonable than trying to encode all receivers and all replication points of a single tree in one packet. However, since this one packet would not reach all receivers, the head end would have to generate as many copies of the data packets as necessary to cover all recipients. As a result this approach would be a compromise between a full tree, and full head end replication. Variations in the size of the 'tree-let' header would allow for space v.s. bandwidth efficiency trade-offs while meeting the goal of remaining stateless in the core.

In the literature there are also non-exact methods to multicast without state such as with Bloom filters [BLOOM]. In this approach the links to be traversed are logically mapped into a field which is carried in the packet (for example if the links are given unique 128 bit sparse addresses then a 128 bit union of all the links to be traversed on the tree is encoded in the header). These mechanisms guarantee that all receivers will get a copy of the packet (because they check each link for inclusion in the Bloom Filter at each hop) however they do so at the expense of sending false positive copies to unintended receivers which must then filter the unwanted packets egress. Depending on the size of the Bloom Filter and the link identifiers various statistical trade-offs in false positives vs. packet header size can be made.

Other exact methods to encode and methods to compute SLSR multicast etc. are FFS.

9. Failure Recovery

A variety of failure recovery techniques can be employed with SLSR. The most obvious is to just re-compute all affected paths on indication of a link failure. This won't be discussed further.

More interesting are the so called fast restoration mechanisms. These can broadly be broken down into head end and tandem restoration.

Head end mechanisms that provide 1+1 protection have been around for a long time with MPLS-TP, PBT and SONET/DWDM. Similar mechanisms can be used with any tunnel type and of course SLSR is no exception. Probes can be sent down one source route, reflected back along the reverse source route and in this manner the forward and reverse paths can be simultaneously probed for failure. In the

event of a failure a diverse alternate source route can rapidly be

added to the packet and the flow restored. The advantage of course with SLSR is that no state is required for either the primary or the backup path. As a result there is little added cost to having even greater redundancy than 1+1 with SLSR. The mechanisms to accomplish this are fairly obvious. Having the reverse path available at the egress means that fate sharing the forward and reverse probes is easy.

In addition to 1+1 protection it is possible to do hop by hop fast reroute type detour protection. This can be done by substitution of a failed link identifier with a set of link identifiers that merge with the path downstream of the failure. An example is given further below for MPLS label stacks, however many other possibilities exist when a history of the packet's path is available to the detour mechanism. The history would permit the detour mechanism to spread the failing packets over different detours and thereby reduce the concentration of additional load imposed by the failure on the same set of links.

10. Comparison of Logical Model to Existing Source Routing

There are a number of existing protocols that support forms of source routing (or can be used to do something close to source routing). IPV4 and V6 had strict and loose node-by-node source routing options (now deprecated) and we'll discuss them briefly. Likewise MPLS behavior can be used to do strict link source routing where a label stack represents a list of link names, this has recently been called segment routing in [SEGMENT].

10.1. MPLS as a SLSR

MPLS is of course not a source routed forwarding protocol, at least not by design. Rather, packets follow an arbitrary path by substitution of a previous hop label with a next hop label and each hop must be pre-configured with the <incoming port, label> to <outgoing port, label> relationship. This is clearly not source routing because tandem configuration is required per path and per hop. However MPLS has a stacking mechanism that can be exploited to create a consumable list of link names to be traversed as they are popped.

The MPLS label stack can therefore be used to implement a flavor of SLSR. This is accomplished by pre-assigning a locally unique MPLS label to each outgoing link of a node. For example in figure 1, node D's link 3 would be assigned MPLS label 3 (but more likely a label value which is 1:1 related to link 3, however we stick with label=link for simplicity of explanation).

The tunnel encapsulation operation would therefore be to push a set of labels onto the frame where each label indicates which link to follow at that given exact strict hop. For example:

SrcIP	DstIP	SrcPrt	DstPrt	MPLS SLSR
192.0.2.4	192.0.2.9	1000	98	push(3,push(2,push(5)))
192.0.2.4	192.0.2.9	1001	99	push(3,push(4,push(5)))

The tunnel tandem operation would then be to pop the label on the incoming frame (after optionally validating its reverse link identifier) and forward to the interface specified by the just popped label value. Every tandem node would be pre-configured approximately as per below. Note that as with any source routing mechanism, this tandem pre-configuration is independent of the actual paths that traverse the node. A table like the one below, with a few hundred interfaces and hence a few hundred labels, could support the transit of an infinite number of TE (or SPF) paths. For clarity we use label N = interface/N but in reality it would be label N = F(interface/N) since a 1:1 mapping 'F' is almost certainly required.

Incoming Interface	Label	Actions
any	1	Pop, forward to interface/1
any	2	Pop, forward to interface/2
:	:	:
any	N	Pop, forward to interface/N

If reverse validation is required the tables would be a bit different because they must match the label to the incoming interface and then pop it and then forward based on the next label. Reverse validation therefore requires two label lookups per forwarding operation.

Finally the tunnel egress operation would be normal forwarding to a VFI or VRF.

MPLS in this manner could be made to do SLSR of unicast frames but cannot be made to reverse the route because the route is consumed in transit. This method also uses many more bits than are really necessary. Each label consumes 32 bits which is rather more than required to express the number of links/adjacencies on a typical switch or router. For example, if the average packet size if 512

bytes, a 5 hop MPLS source route imposes a 4% overhead (20/512) on

some links with the largest overhead on the first few links. For larger packets this is likely not an issue, for smaller packets it is possibly a concern.

A more realistic number actually required per hop is probably 8 or 12 bits (256-4K links) and if more bits are required two hops can be consumed by any node with such a large nodal degree. The MPLS label also has an 8 bit TTL which is of course redundant in any source routing mechanism. This begs the question of if a smaller MPLS label would not be more suitable?

There are other issues with the use of MPLS, in particular current hardware can usually not stack very many labels at a time (3 on some popular ASICs). This would limit the network diameter to 4 hops. Of course NPUs or new ASICs could be extended to allow further ingress stacking.

It does not seem possible to do SLSR multicast with MPLS except of course via head end replication.

The hop(ingress stack size) limit, lack of reverse, consumable route and lack of efficient multicast still do not invalidate use of MPLS source routing for many networks and its use would have a noticeable positive impact on the scale/speed of a central controller in such environments.

MPLS fast reroute mechanisms can also be implemented locally in a similar fashion thus further improving controller scale by alleviating the need for 50ms responses network wide from the controller and giving the controller more lee-way to recover after the fast reroutes have detoured traffic around the failed nodes and/or links.

Consider possible local actions when the link A.2 between nodes A and B in Figure 1 fails. Since there is still a link A.4 available, the node A can locally change the action associated with label 2 to instead send to interface 4 when interface 2 fails. If an entire adjacency fails, such as would happen when both A.2 and A.4 fail, then a link detour can be locally performed by reprogramming the actions for labels 2 and 4 to now push labels 3,3 and send to interface 3. This will cause a detour via D back to B. More elaborate kinds of detour are possible by processing two link names ahead instead of one, including nodal detours. These can be done locally without end to end path knowledge and hence scale independently to the number of paths. Eventually the controller will detect the failure and reconstruct the SLSRs at the head end and the use of the detour will stop without having to withdraw any state in the core.

If MPLS is of use in the context of SLSR then it would be worth considering a number of future extensions to MPLS. Some things to consider could be a smaller MPLS label option, say 16 bits with no TTL and the possibility of not popping but rotating the label to the bottom of the stack to preserve the path history for OAM and reversibility reasons. While these sorts of things are of course not possible with existing ASICs they are easy to do on existing NPU's and new work on Protocol Oblivious Forwarding [POF] allows near arbitrary bit pattern/action matches to be programmed by an SDN controller permitting a more optimal data path encoding of SLSR than can be obtained by simply reusing MPLS.

10.2. IPV4/6 Options as SLSR

IP header option 9 [RFC791] defined (but now deprecated) the Strict Source and Record Route (SSRR) option for IPV4 packets. This option has(had) a 'length' field, a 'pointer' field and an array of 'route data' fields. The element in the array of 'route data' indexed by the 'pointer' field contains the IP address of the immediate next hop towards which the packet must be forwarded, the 'pointer' field is incremented, and the previous hop is filled in with the IP address of the current device prior to actually forwarding the packet. Up to 9 hops could be specified in this manner. IPV6 also had a similar option "RH0" which is also now deprecated [SRBAD].

IPV4 and V6 Strict Source and Record Route methods could be used to implement Strict Link Source Routing. This would be accomplished by assigning a 32 bit number to the link and then using the 32 bit number in place of the IPV4 or V6 address in the route list.

In both IPV4 and IPV6 the source routing options were found to be harmful to the Internet at large for a number of reasons. These reasons are described in [SRBAD] but briefly there were two broad classes of problem encountered. 1) Harm to intermediate links and 2) harm to end hosts. For example:

- Since it was possible to list a waypoint more than once in the route data, it was possible to loop traffic around multiple times (9 times in the case of IPV4 and 90 times in the case of IPV6). This looping allowed saturation of high speed links by hosts that had an order (or two) smaller bandwidth access to the Internet. A congestion style DOS was therefore possible from low speed access links against higher speed core links.

- Various schemes such as bypassing of firewalls etc. are of course easy to do when a host can specify waypoints that detour

around a firewall.

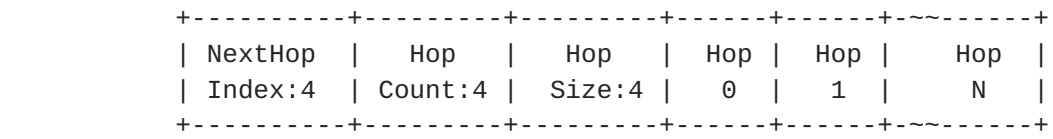
- Spoofing using the reverse route. Since the reverse source route is installed against the IP SA by a host that receives it, it is possible to use a bogus IP SA in combination with a reverse source route that detours the packets to the imposter host.

10.3. Protocol Oblivious Forwarding as SLSR mechanism

The OpenFlow [OPENFLOW] protocol defines methods for an external controller to cause the manipulation of known packet headers and fields within those headers by a forwarding element. As such it is currently limited to matching on known fields like MPLS, IP, Ethernet etc. and taking actions on those fields. While flexible there are still many things at the data path level that OpenFlow cannot do including generic source routing such as SLSR.

The Protocol Oblivious Forwarding [POF] protocol is a proposed extension to OpenFlow which permits arbitrary bit pattern matching/actions and is therefore much more flexible. The goal of POF is to allow a controller to define a new data path in addition to a new control plane and to then program the data path on the forwarding elements to its specifications. POF is therefore not limited to existing IP, MPLS, Ethernet fields.

It would therefore be possible with POF to implement a highly flexible SDN tunnel data plane that closely resembles the idealized SLSR data path. Strictly by way of example POF could implement a flexible SLSR header along the following lines:



With only five bytes, this header could represent 3 hops with 256 links per hop, 4 hops with 64 links per hop, or 6 hops with 16 links per hop, etc. With additional bytes of course more/longer combinations are possible with very reasonable overhead. This is considerably more compact than the other described options and without sacrificing reversibility or giving up the OAM benefits of knowing the exact path the packet has taken.

POF however could also implement other variations of SLSR based on MPLS. For example POF could implement a smaller MPLS label, say a 16 bit label without a TTL. POF could theoretically also implement a rotating label list instead of a popping label stack.

POF appears to be ideally suited for SLSR developments beyond what can currently be done with MPLS.

11. Security Considerations

Source Routing security concerns are also discussed in the previous section related to IPV4 and IPV6 now deprecated nodal source routing.

This draft is proposing link based source routing and that it be used as a tunneling mechanism only. This means that only devices that are at the edge of an SDN sub-network would be allowed to insert strict link source routes. Note that an MPLS label can only be inserted by a Label Edge Router (LER) and processed by Label Switch Routers (LSR) and not by end hosts. Therefore SLSR should be no more or less secure than MPLS. In fact the absence of signaling protocols like RSVP-TE removes a point of attack. The fact that this mechanism is intended for use by a central controller further mitigates the possible attacks as encrypted communications are used to the edge devices which are the only device able to insert the strict link source routes.

There is however the possibility that an attacker could attach to a core device and inject strict link source routed packets. Methods to prevent this however are not hard, in particular the adjacency would have to be reported to the controller and the controller would have to enable packet forwarding. Unless the controller recognized both ends of the link as being part of its controlled domain it should not enable the strict link source routing capability on that interface thus preventing the threat.

Other interfaces, such as those facing a network of hosts or devices not in the domain of the controller would, as with current BCP's, drop any source routed frame in any format (new or old).

As previously mentioned there are ways to spread the link names into a 32 bit space such that the exact mappings are only known by the controller and the tandem node in question. This would prevent any easy form of guessing being used to construct an SLSR. One such example of this kind of secure source routing is given in [SANE].

Source Routing also is unique in that the packets themselves give details about slices/cuts through the topology, therefore with sufficient interception of packets from diverse sources and destinations in the network, an attacker could build up a detailed view of the network topology, this would be a concern for a carrier SDN network in particular where details of topology are considered a valuable asset, although exploiting knowledge of the topology would be more challenging given the secure protocols that exist between a controller and the forwarding entities.

In the SDN context there appears to be little need for a loose source route. Loose source routing adds additional security concerns because it does not require knowledge of the entire path to construct an attack. If loose source routing is included the security concerns should be addressed.

12. Conclusions and Future work

SDN where a central controller creates either pro-actively or re-actively the state for a sub-network of forwarding devices will have performance limitations that are related to network diameter/size, network recovery requirements and the amount of state they need to distribute. Strict Link Source Routing mechanisms can alleviate these problems allowing greater scale and faster recovery. MPLS can be used to implement this on a small scale with some of the benefits. IPV4 and IPV6 source routing options can be used to implement this on a larger scale with more of the benefits but at much larger packet overhead but are however perceived as risky and have been deprecated from IP. These risks however can be mitigated in this specific use. No existing mechanism however is optimum, and therefore there is room for a new mechanism that addresses these requirements and includes multicast methods and more efficient encoding of link names than is currently possible. One possible solution is to look at a smaller MPLS label for this purpose and to look at ways to retain the popped labels for the purposes of end to end path reversal and OAM. New work in SDN, in particular Protocol Oblivious Forwarding may make these kinds of things possible in a generic manner.

13. IANA Considerations

This memo includes no request to IANA.

14. References

14.1. Informative References

[BLOOM] Active Bloom Filters for Multicast Addressing, Z. Heszberger et. al. Budapest University of Technology and Economics.

[OPENFLOW] www.openflow.org

[ONF] www.opennetworking.org

[POF] Protocol Oblivious Forwarding: http://www.poforwarding.org/

[PLACEMENT] The Controller Placement Problem, Nick McKeon, Brandon Heller, Rob Sherwood. HotSDN'12, August 13, 2012, Helsinki, Finland. 2012 ACM 978-1-4503-1477-0/12/08, http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p7.pdf

[RFC791] Internet Protocol, Information Sciences Institute, RFC 791, September 1981.

[SANE] SANE: A Protection Architecture for Enterprise Networks, Martin Casado, Nick McKeown, http://yuba.stanford.edu/~casado/sane.pdf, Stanford and ICSI 2005.

[SDNGOOG] SDN at Google - Opportunities for WAN optimization, E. Crabbe, V. Valancius, 8/1/2012. Presentation at IETF84 SDN BOF.

[SEGMENT] Segment Routing with IS-IS, S.Previdi et. Al. http://tools.ietf.org/html/draft-previdi-filsfils-isis-segment-routing-00

[SLSR] Software Defined Networking and Centralized Controller State Distribution Reduction, www.ieee802.org/1/files/public/docs2012/new-ashwood-sdn-optimizations-0712-v01.pdf

[SRBAD] Deprecation of Source Routing Options in IPV4 http://tools.ietf.org/html/draft-reitzel-ipv4-source-routing-is-evil-00

[SRSDN] Source Routed Forwarding with SDN, M. Soliman http://conferences.sigcomm.org/co-next/2012/eproceedings/student/p43.pdf

15. Authors' Addresses

Peter Ashwood-Smith
Huawei Canada Inc.
303 Terry Fox Drive, Suite 400, Kanata, Ontario K2K 3J1
Email: Peter.AshwoodSmith@huawei.com

Mourad Soliman
Carleton University,
1125 Colonel By Drive Ottawa, Ontario K1S 5B6 Canada
Email: MouradSoliman@cmail.carleton.ca

Tao Wan
Huawei Canada Inc.
303 Terry Fox Drive, Suite 400, Kanata, Ontario K2K 3J1
Email: Tao.Wan@huawei.com

16. Contributors

We invite more contributors.

17. Acknowledgements

We gratefully appreciate the feedback of Nigel Bragg, Sue Hares, Peter Willis, Biswajit Nandy and Linda Dunbar.