Towards Reliable and Lightweight Source Switching for Datacenter Networks

Feng Wang[†], Lixin Gao[*], Shao Xiaozhe[*], Hiroaki Harai[§], Kenji Fujikawa[§]
[†] School of Engineering and Computational Science, Liberty University, Lynchburg, VA 24515, USA
[*] Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA
[§] National Institute of Information and Communications Technology, Tokyo 184-8795, Japan

***Abstract*—A low-latency and reliable message switching network is critical for constructing high-speed datacenter networks. In this paper, we present the design, implementation, and evaluation of a novel Location basEd Source Switching (LESS) for datacenter networks. LESS enables lightweight source switching through a location-based addressing scheme. Each switch and host can independently derive a source route to reach a destination without requiring the full knowledge of the network topology. We demonstrate that using location-based source routes as forwarding labels allows LESS to eliminate the need for routing tables and integrate with minimum required functionality for packet forwarding. Moreover, we propose a fast rerouting solution to address the issue of fault tolerance in source routing. Each switch can locally derive an alternative source route during a failure. The paper evaluates the performance of LESS. Our evaluation results suggest that LESS improves the performance of datacenter networks in terms of latency, throughput, and reliability.**

I. Introduction

As datacenter network demands have increased, designing and implementing a scalable, reliable and high performance datacenter network becomes essential. Routing is a key design issue of datacenter networks. Recently, two routing strategies have been adopted. Centralized flow routing in SDN technology [1], [2] relies on a central controller to provide flexibility and improve the bandwidth efficiency. However, centralized solutions, such as OpenFlow based switches, are too complicated and are difficult to maintain [3]. Another scaling constraint is the size of forwarding tables as the table size grows linearly with network size [4]–[6]. More importantly, it usually takes substantial time for a centralized controller to update switches to reroute around failures [7]–[9].

On the other hand, source routing has been proposed for use in datacenter networks. A source route encodes a path as a sequence of identifiers of the intermediate hops along the path. The source node computes the path and stores it in the packet header. As a result, routers are very simple and fast. Recently, SDN-based source routing has been proposed [10]–[12]. A centralized SDN controller assigns each switch's outgoing interface a unique binary number, and then installs a flow-table in each edge switch. However, the overhead of implementing source routing is still high because full topology information is required to compute the source routes. For example, in work [3], hosts need to send probes to obtain full topology information to compute the source routes. Furthermore, those approaches require a lot of administrative effort to make sure that each switch maintains per flow states. For instance, an Openflow switch needs to maintain entries for packet flows in flow tables and perform a table lookup to translate an identifier to an interface.

In this paper, we design and implement a new Location basEd Source Switching (LESS) system. Inspired by previous work [3], [11], [13], LESS is designed to enable packet forwarding via source switching, in which a source route is used to specify output ports that packets should traverse. Compared with Openflow-based switches, LESS provides the flexibility of source routing without its overhead. First, each switch in LESS is a routing-tableless implementation integrating the minimum required functionality for packet forwarding. Since the output ports at each hop are embedded in packet headers, LESS eliminates expensive lookup tables and label translation operation at switches. Second, LESS addresses the overhead of network topology discovery in source routing by providing a scalable addressing scheme. Moreover, LESS simplifies the deployment and management of a large scale datacenter network. Only hosts perform a mapping from IP addresses to the proposed pseudo MAC addresses. Contrastingly, most existing approaches must perform a similar mapping at each host and switch, and install new routes at switches by a centralized controller.

More specifically, LESS employs a distributed address configuration scheme to assign Location-based pseudo MAC addresses to switches and hosts. Each address is a sequence of input and output ports in each of the succession of switches from a root switch to a switch or host. Since a switch or host may have multiple addresses, the addresses can be used to efficiently discover the network topology. An algorithm is proposed to calculate a source route based on the addresses. A dynamic local rerouting solution is then used to provide uninterrupted communication during link failures. We have implemented a prototype of LESS in both hardware and software. The measurements demonstrate that LESS can enhance the performance of datacenter networks in terms of packet processing latency, end-to-end delay, throughput and reliability.

The rest of the paper is organized as follows. In Section II, we introduce the architecture of LESS. We present the Location-based MAC addressing scheme in Section III, and LESS operation in Section IV. A fast failover solution is proposed in Section V. We evaluate the performance of

978-1-5090-5336-0/17/$31.00 ©2017 IEEE

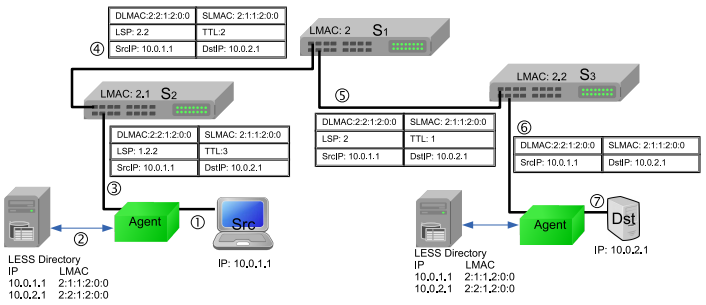
the proposed encoding scheme in Section VI. In Section VII, we present the related work. We conclude the paper in Section VIII with a summary.

II. LESS Architecture

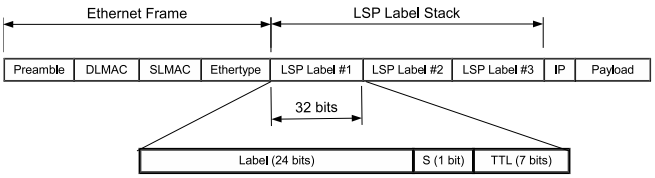
Fig.1(a) shows the system diagram of LESS, which consists of four main components: *a hierarchical addressing scheme*, *an LMAC directory system*, *LESS agents*, and *LESS-enabled switches*. The hierarchical addressing scheme is employed to assign location-based addresses to switches and hosts (Section III). This dynamically-assigned address is referred to as a *Location-based* pseudo MAC (LMAC) address. LESS uses a directory system to maintain the mappings between IP addresses and LMAC addresses. Like the pseudo MAC address proposed in PortLand [14], LMAC addresses are based on the MAC rewriting technique. A LESS agent running on every host invokes the directory system's resolution service to select a pair of source and destination LMAC addresses to reach a destination. However, only LESS agents need to maintain a one-to-one mapping between LMAC addresses and hosts. Each agent replaces the source and destination MAC addresses with the LMAC addresses so that switches do not need to rewrite MAC addresses with the pseudo MAC. Based on a pair of LMAC addresses, an algorithm is proposed for host agents to derive a source route to reach the destination (Section IV). Since LESS moves the source route determination to end hosts, LESS-enabled switches are simplified to provide high performance and reliable forwarding: (1) forwarding packets to the next hop according to a pre-configured source route; and (2) locally deriving an alternative source route to reroute around a link failure (Section V).

To be more specific, every agent performs LMAC address resolution and source route selection (Step ② in Fig.1(a)). A LESS agent replaces Ethernet ARP queries with the LESS directory system queries to resolve an address. It translates a destination IP address into a set of destination LMAC addresses. Various selection methods can be used to select the LMAC addresses. The translation occurs transparently so that hosts can direct connect to a LESS agent without modification, and the applications are unaware of LESS. Once a pair of source and destination LMAC addresses is determined, an agent assembles a source route based on the addresses and uses it for the subsequent packets. The source and destination agents keep track of active flows and their chosen LMAC addresses.

LESS uses a novel source-switching protocol to transfer data. Each source route is encoded into source switching labels, which are implemented as a shim layer between the link layer and IP layer. Fig.1 (b) shows an example of two source switching labels. A LESS packet header contains 1) an Ethernet frame, 2) a set of source switching labels, 3) an IP header and 4) a payload. A source switching label includes a sequence of output ports in successive switches through which the packet is to be transmitted. The detail of source switching labels is presented in Section IV.



(a) LESS Architecture



(b) Packet Header Format

Fig. 1. LESS architecture and LESS packet header format.

Each LESS switch forwards packets based on the source switching labels. The labels are pushed and popped on and off packets as they flow in a LESS switch. As a packet passes through a switch, the switch strips the top label to derive the output port through which the packet will be transmitted. For instance, a packet with a switching label "`1.2.2`" is transmitted to switch S_2 (Step ③ in Fig. 1(a)). The first number of the switching label is extracted to determine the outgoing port (port 1) for the packet. Then, the number is deleted from the label, and the label becomes "`2.2`". Meanwhile, switch S_2 decrements TTL value in the label. Finally, the switch sends the packet to the next switch S_1, which has a physical link to the outgoing port 1 (Step ④). The process will continue until reaching the destination host. At switch S_1, the output port is the first number of label "`2.2`". The switching label sent by S_1 becomes "`2`", and the TTL value is 1 (Step ⑤). Upon receiving the packet, switch S_3 extracts the output port 2. Since the TTL value becomes 0 after decremented by 1, the whole switching label is deleted. Finally, the destination host receives a regular IP packet (Step ⑥). Consequently, the destination host can generate a switching label for return packets based on the source and destination LMACs carried in the incoming packets.

III. Location-based MAC Addressing

In this section, we present an LMAC addressing scheme for a datacenter network with a generalized multi-rooted hierarchical tree topology. This addressing scheme can allocate LMAC addresses dynamically and automatically.

A. Datacenter Topologies

Many datacenter architectures follow multi-rooted hierarchical trees [14]–[16], for example, fat-tree [15] and Clos [17]

topologies. Multi-rooted hierarchical trees are characterized by providing multiple parallel paths between end hosts to support full bisection bandwidth and enhance the performance of the datacenter. In a multi-rooted tree topology, LESS switches are partitioned into levels. Each switch belongs to exactly one level, and connects to switches in the above level or below it. The root switches are at level 0, and only have children. Hosts are at the lowest level, and only have parents. Other switches have both parents and children. The generalized multi-rooted tree topology is flexible. For example, each switch may have a different number of ports. A pair of switches at the same level may connect to each other via a peer link. Moreover, the network topology can be modified after the network is deployed. For ease of exposition, we will use a fat-tree topology for our addressing scheme description and experimental evaluation because it represents a typical hierarchical datacenter network.

B. LMAC Assignment

An LMAC address is represented as a fixed six-byte identifier, as illustrated in Fig. 2. Each LMAC address represents a location label of a switch or host with respect to a root switch. Particularly, each root switch is pre-assigned a distinct location label. For example, the root switch in Fig. 2 is assigned a unique label "`1:0:0:0:0:0`". To simplify the representation of an LMAC address, we ignore all 0s after the label. Then, the root switches begin by assigning LMAC addresses for their children. All other switches and hosts inherit LMAC addresses from an upper switch. When assigning LMAC addresses, only the switch having at least one LMAC address can allocate a new LMAC to another switch.

Thus, an LMAC address can be interpreted as a downward path from a root switch to a switch or a host. Each switch along the path is represented by a pair of switch-local port numbers. We define two types of switch ports: *upward* port and *downward* port. In a multiple-root topology, upward ports are connected to their parents, while downward ports are connected to children. In addition, root switches only have downward ports.

Formally, upon receiving an LMAC address l, switch A would assign a new LMAC address l' to another switch B by prepending label l with the upward port from which l is received, and the downward port to which switch B is connected. Or, $l' = l : i : j$, where i and j are the upward and downward ports, respectively. Since root switches do not have upward ports, they advertise their location labels by pre-pending their downward ports only.

For example, in Fig. 2, the root switch would pass its location label "`1:`" to switch 1 by pre-pending the downward port number "`1`". As a result, the LMAC of switch 1 is "`1:1:`". After that, switch 1 could allocate a new LMAC to switch 3 by prepending its LMAC "`1:1:`" with the upward port number "`1`" and the downward port number "`2`". Thus, the LMAC of switch 3 is "`1:1:1:2:`". Similarly, host 2's LMAC address is "`1:1:1:2:3:8`" by pre-pending switch 3's ports. In LESS, a host could connect to more than one edge switch via multiple ports.

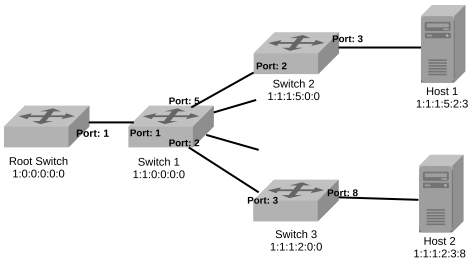


Fig. 2. Assignment of LESS addresses by switches.

It is important to note that every port number is locally assigned. When two switches are directly connected via a link, they assign the port numbers locally without negotiating a link label. This is the key difference between previous work. For example, in work [13], switches need to negotiate a distinct number for each link.

To avoid a possible loop during LMAC assignments, each switch deletes all LMAC addresses for which it has another LMAC that is a common prefix of those addresses. For example, suppose that switch A might accidentally pass "`1:1:1:3:1:3`" to another switch B. In this case, B would delete the LMAC if it already has another LMAC "`1:1`" because "`1:1`" is a prefix of "`1:1:1:3:1:3`". Furthermore, a strict policy can be employed to avoid a possible LMAC assignment loop: *each switch would not pass an LMAC address via its upward ports*.

C. LMAC Encoding

As shown in Fig 1(b), each LMAC is encoded into a 48-bit MAC address. We use one byte to represent a root switch label and a port number. As a result, a datacenter network could have a maximum of 255 root switches. Each switch can have at most 255 upward ports and 255 downward ports[1]. For a fat-tree topology, 48 bits are large enough to encode all the LMAC addresses.

D. LMAC Assignment Protocol

An LMAC assignment protocol is designed to assign, maintain and update LMAC addresses. Due to the space limitation, we briefly describe the protocol. First, a set of switches are selected as the root switches, and each of them is allocated a location label. Then, every non-root switch sends local multicast messages within a one-hop scope to its adjacent switches requesting LMAC addresses. Upon receiving a response message, root switches, or the switches with assigned LMAC addresses, reply to the request with one or multiple LMAC addresses. A switch could use either all assigned LMAC addresses or a subset of addresses. The process is repeated recursively until all the switches/hosts acquire a set of LMAC addresses.

After a switch acquires its LMAC addresses, it periodically sends an update message to maintain the assigned LMACs. If a parent switch does not receive the message from its child node within the specified time, it considers that the child switch has

[1]Port 0 is reserved for the control plane.

failed or moved out of the datacenter network. It will set the downward port to the child switch to unavailable. Similarly, if a child switch does not receive the update message from its parent switch within the specified time, it considers that the parent switch has failed or moved out of the network. In this case, it will delete all LMAC addresses received from the parent switch. Finally, it broadcasts a message via its downward ports to withdraw the failed LMAC addresses. Upon receiving the message, each of its child switches will delete the corresponding LMAC addresses, and broadcast a withdrawal message to its child switch. The process is repeated until all the switches/hosts update their LMAC addresses. When a new switch joins a multi-root tree, it broadcasts its requests to its parent switches to request LMAC addresses. After a parent switch receives the request, it returns LMACs to the new switch.

E. LMAC Selection

The number of LMACs bound to a host will explode with the number of levels and ports. Therefore, it is not practical for hosts to store all possible LMAC addresses. If a switch only notifies its neighbors of a fixed number of LMAC addresses, the number of LMACs can be minimized. Therefore, every switch should be instructed to choose only the best or a subset of LMAC addresses, and filter LMACs that are less likely to be used. For example, a switch can limit the number of LMACs bound to hosts or switches by passing k LMACs with the lowest latency to reach the root switches. We may select the LMACs that are disjointed from each other to make failure recovery easier. Different LMAC addresses might be used for different applications running at the same host. Thus, packets belonging to different applications can travel along different paths. In addition, we may use load or latency as inputs to the LMAC selection. We can pick a pair of LMACs that have the lowest RTT or loss rate. However, most importantly, in order to guarantee the reachability of hosts, *every host must have at least one LMAC address inherited from the same root switch*.

F. Elongated LMAC Addresses

The length of an LMAC address is not limited by 48 bits. LESS can be applied to a large scale datacenter network, for example, a fat-tree topology with more than 3 levels. Instead of encoding an LMAC address into the Ethernet MAC address fields, the elongated LMAC addresses can be encoded into an IP packet header, such as IP Option field. Note that the location of LMAC addresses in packet headers does not impact LESS packet forwarding because the LMAC addresses are used only at end-hosts or intermediated switches when a link failure occurs.

IV. Port Based Source Switching

We can utilize LMAC addresses to build a source route, which is defined as a *Label Switched Path* (LSP). An LSP consists of a sequence of egress port numbers that should be followed for subsequent switches along the path from source to destination.

Algorithm 1: *LSP Construction*

input : source LMAC M_s and destination LMAC M_d.
output : LSP L

Calculate $C = M_s.commonPrefix(M_d)$;
if $C == \emptyset$ **then**
  return $L = \emptyset$;
end
Remove the common prefix,$M'_s = M_s.remove(C)$, $M'_d = M_d.remove(C)$;
Reverse $S = M'_s.reverse()$;
Concatenate S and M'_d, $Q = S \odot M'_d$;
Generate L by selecting the numbers at the even positions, $L = Q.select()$;
return L

A. LSP Construction

When a host starts data communication, it obtains the destination LMAC addresses by resolving the destination IP address from the LESS directory system. The directory system supports a DNS-like hierarchical directory service. After the host obtains the destination LMAC addresses, it initiates the LSP construction process. An LSP to an intended destination is obtained by *concatenating* the source and destination LMAC addresses. In order to concatenate the two addresses, they must share a common prefix. Thus, an LSP consists of an ascending path to one of the nearest common ancestors of the source and destination switches and then a descending path to the destination. Formally, in order to build an LSP from a host with LMAC $p_{n-1} : \ldots : p_1 : p_0$ to a destination with LMAC $p'_{n-1} : \ldots : p'_1 : p'_0$, the two LMAC addresses must have a common prefix, $p_i = p'_i$ for $i \in \{n-1 \ldots j+1\}$ and $p_j \neq p'_j$.

Algorithm 1 is presented to build an LSP. In this algorithm, $commonprefix()$ function is used to derive the common prefix part of two LMAC addresses. After removing the common prefix from the LMACs, the source LMAC is reversed, denoted by $reverse()$. The two addresses are then concatenated. Symbol $\odot$ represents the concatenation operator. Finally, we select the numbers at the even positions to generate the LSP.

For example, in Fig. 2, suppose that host 1 sends a packet to host 2. By comparing the LMACs of the two hosts, there is one common prefix: "`1:1:1:`". After removing the common prefix, we reverse the pruned source LMAC: "`2:3:8:`". Next, the LMAC is combined with the pruned destination LMAC: "`5:2:3`", and the label becomes "`8.3.2.5.2.3`". Finally, we select the port number at the even position to generate LSP "`3.5.3`".

B. LSP Encoding

LSPs are carried in a shim header between the data link layer and IPv4 layer. The length of each LSP label is 32 bits, and more than one label can be encoded into a packet header. The LSP label format is shown in Fig. 1(b). We leverage a Multi-Protocol Label Switching (MPLS) encapsulation format to represent LSP labels. The first 24 bits encode 3 port numbers. The next bit (S bit) is a label stack bit, which indicates the last LSP label. If the stack bit is 1, it indicates

TABLE I
LSP REWRITE ACTIONS FOR DIFFERENT LSP LABELS

LSP Type	LSP Rewrite Actions
Last Hop	1) drop the last LSP label 2) modify Ethernet type to 0x0800
Last LSP	1) left shift the first 24 bits of the last LSP by one byte 2) rewrite a new TTL value
Last Port	1) drop the top LSP label
Top LSP	1) left shift the first 24 bits of the top LSP by one byte 2) rewrite a new TTL value

that the LSP label is the last label, and an IP packet header is stored after the label. Otherwise, there is another LSP label before the IP packet. The next 7 bits are the Time To Live (TTL) field, which is decremented by 1 at each hop. If the TTL reaches 0, the LSP label is dropped.

C. Source Switching Operation

For an incoming packet, we define four LSP label types:

- **Last Hop**: the first LSP label is the last one with a TTL value of 1.
- **Last LSP**: the first LSP label is the last one with a TTL value larger than 1.
- **Last Port**: there are more than one LSP label, and the first LSP label has a TTL value of 1.
- **Top LSP**: there are more than one LSP label, and the first LSP label has a TTL value larger than 1.

Based on the LSP type of incoming packets, a switch performs the following LSP label manipulation actions: 1) decrement the TTL, 2) left shift the first 24 bits of the top LSP by one byte, 3) rewrite a new TTL value, 4) drop the last or top LSP label, and 5) modify Ethernet type. Table IV-C shows the rewrite actions for every type of LSP label after decrementing the TTL value. The operations are repeated for every switch through which the packet traverses. When a packet reaches its destination, the entire LSP labels have been removed from the packet so that the destination host only obtains an ordinary IP packet.

V. DYNAMIC FAULT TOLERANCE WITH SOURCE SWITCHING

One of challenges in source switching is handling a link failure. In this section, we present a dynamic fault tolerance solution.

A. Alternative LMAC Addresses

A host can have multiple LMAC addresses, and each of the addresses represents a unique path to the host. This allows switches to reroute packets via alternate paths in the event of a link failure. In order to use an alternative path against a link failure, a pair of alternative source and destination LMAC addresses should be encoded into packet headers. We propose to encode them into the IP Option field, and Loose Source Route could be used as the Option type.

In general, when a link between two switches fails, both switches would disable the ports. Upon receiving a packet, each switch should check if the port from the top LSP label (hereinafter referred to as *primary* LSP label) is to the failed

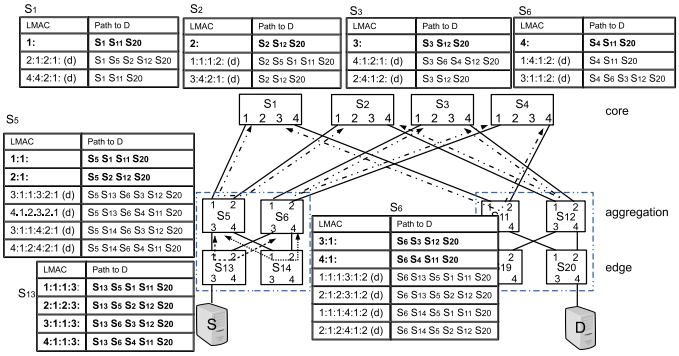


Fig. 3. The number of LMAC addresses and path diversity at core, aggregation and edge switches after employing LMAC deflection. A table beside a node denotes the LMAC addresses and the corresponding path (switch level) to reach a destination. The local LMAC addresses are in bold font, the deflected LMAC addresses are indicated by Letter 'd'.

link. If this is true, the switch inspects the IP option of the packet. If the packet does not carry alternative addresses, the switch drops the packet. Otherwise, the switch continues to examine if both alternative source and destination LMAC addresses utilize the failed link. If this is true, the packet is dropped because the alternative path is affected by the failure as well. Otherwise, the switch will use the alternative LMACs to compute a new LSP, which is referred to as a *rerouting LSP*. Then, the primary LSP label is replaced by the rerouting LSP. Meanwhile, the primary source and destination LMAC addresses at the Ethernet address fields are replaced by the alternative LMACs. Note that once the rerouting LSP is calculated, the subsequent packets are detoured to the destination by using the label.

B. LMAC Deflection

It is desirable to have highly redundant paths between hosts to achieve fast failover. However, in a fat-tree topology, only the edge switches have high path diversity. The aggregation and core switches have less number of paths to reach the same destination. Specifically, each root switch has only one single path to reach any host. Fig. 3 shows the number of LMAC addresses and the corresponding paths to the same destination. The figure only depicts parts of the fat-tree topology.

To increase path diversity, we propose to let aggregation and edge switches deflect LMAC addresses to their upper level switches via upward ports. In Section III, switches can broadcast their LMAC addresses only to their child switches via downward port. In other words, each switch is expected to receive LMAC addresses from its upward ports, instead of downward ports. Here, we release this restriction. That is, a switch might receive an LMAC address from its downward ports. In this case, an LMAC address received from a downward port is defined as a *deflected LMAC* address. The original LMAC addresses are called *local* addresses to differ from deflected LMAC addresses. To avoid a possible LMAC assignment loop, when receiving a deflected LMAC, the upper level switch would use it locally and is not allowed to broadcast it to any other switches. In Fig. 3, the edge switches and aggregation switches deflect their LMAC addresses to

Algorithm 2: *Rerouting LSP Construction*

input : primary source LMAC M_s, alternative source and destination LMACs M'_s and M'_d, and switch u's LMACs ω.
output : The corresponding rerouting LSP L_α

```
foreach i ∈ ω do
    C = M'_d.commonprefix(i) ;
    C_max = max(C) ;
    k = i with max(C) ;
end
if C_max then
    /* reroute directly to the destination edge */
    return L_α = Algorithm 1 (M'_d, k) ;
else                          /* reroute via the source edge */
    foreach i ∈ ω do
        C = M_s.commonprefix(i) ;
        if C ≠ ∅ then
            k = i ;
        end
    end
    L_back = Algorithm 1 (k, M_s) ;
    if u is a root switch then remove the first byte from L_back ;
    remove the last byte from L_back ;
    L_alt = Algorithm 1 (M'_d, k) ;
    return L_α = L_back ⊙ L_alt ;
end
```

TABLE II
SWITCH'S OWN LMACS USED FOR REROUTING LSP CONSTRUCTION AND THE LENGTH OF GENERATED REROUTING LSPS

Switch Layer	Unavailable Port	Switch's LMACs	Length (hop count)
Root	downward	deflected LMAC	9(4)
Aggregation	upward	local LMAC	5(0)
		deflected LMAC	7(2)
	downward	deflected LMAC	7(2)
Edge	upward	local LMAC	5(0)

upper level switches. From this figure, we find that only core and aggregation switches have deflected LMACs. That means, LMAC deflection does not increase the number of LMACs at end hosts and edge switches.

C. Rerouting LSP Construction

A link failure might occur between a root switch and an aggregation switch, or between an aggregation switch and an edge switch. We do not consider a link failure between a host and its edge switch because there is only one link between them. In addition to the alternative LMACs carried in packet headers, a switch uses one of its local LMAC addresses or deflected LMAC addresses to construct the rerouting LSP. Table II shows the cases when a local or deflected LMAC address should be used. Algorithm 2 shows the steps to compute the rerouting LSP. The rerouting LSP might be longer than the primary LSP. Table II shows the length of a rerouting LSP built by different switches. The number in the parenthesis indicates the incremented hop count. In the following discussion, we use three examples to illustrate the rerouting LSP construction performed at different switches.

Edge Switch: the edge switch's local LMACs and the alternative destination LMAC are used to derive a rerouting LSP. The rerouting LSP has the same length as the primary LSP. For instance, suppose the link between S_5 and S_{13} in Fig. 4 fails. Edge switch S_{13} compares its local LMACs with the alternative destination LMAC "`3:4:2:4:2:4`". One of addresses "`3:1:1:3:`" has a common prefix with the alternative LMAC. After removing the common prefix "`3:`" from the two LMACs, they become "`1:1:3`" and "`4:2:4:2:4`", respectively. Next, LMAC "`1:1:3`" is reversed and then concatenated by LMAC "`4:2:4:2:4`". The string becomes "`3.1.1.4.2.4.2.4`". Finally, the numbers at the even positions are used to generate the corresponding rerouting LSP "`1.4.4.4`", which represents the output port at switch S_6, S_3, S_{12}, and S_{20}, respectively.

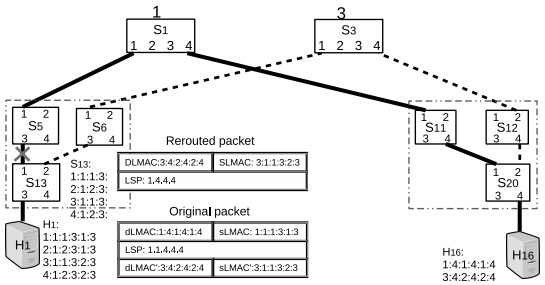


Fig. 4. An example of a rerouting LSPs constructed by edge switch S_{13} to reroute packets sent by host H_1 to H_{16} during a link failure between S_5 and S_{13}. Bold lines represent the paths used before the link failure, and dashed lines represent the rerouting path during the failure.

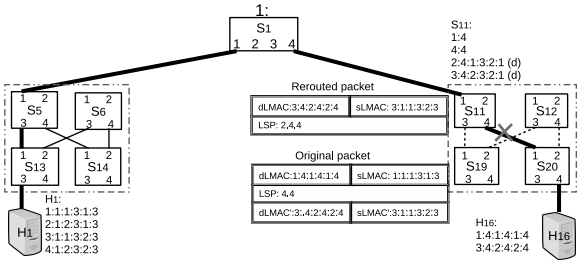


Fig. 5. An example of a rerouting LSP constructed by aggregation switch S_{11} to reroute packet sent by host H_1 to H_{16} during a link failure between S_{11} and S_{20}.

Aggregation Switch: in this case, if the unavailable port is a downward port, the switch's deflected LMAC and the alternative destination LMAC are used to build the rerouting LSP. The length of the rerouting LSP is two hops longer than the length of the primary LSP. Otherwise, if the upward port is down, the switch's local LMAC or deflected LMAC might be used with the alternative destination LMAC to build the rerouting LSP, which might be longer than the primary LSP if a deflected LMAC is used. For example, suppose that the link between S_{11} and S_{20} is down in Fig. 5. One of the deflected LMACs "`3:4:2:3:2:1`" has a common prefix with the alternative destination "`3:4:2:4:2:4`". After removing the common prefix, the two LMACs become "`3:2:1`" and "`4:2:4`", respectively. LMAC "`3:2:1`" is reversed, and then combined with "`4:2:4`". By selecting the even position numbers from "`1.2.3.4.2.4`", the reroute LSP label is "`2.4.4`".

Root Switch: the root switch cannot find any LMAC address having a common prefix with the alternative destination LMAC. It has to reroute the packets via the source edge switch. To do that, the switch first determines the rerouting

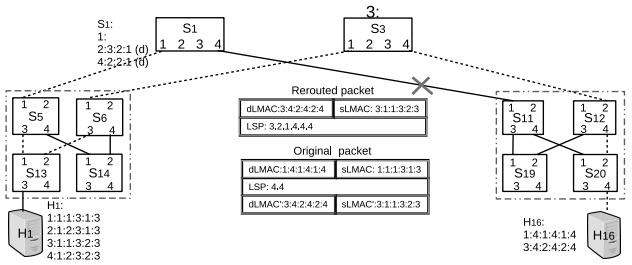
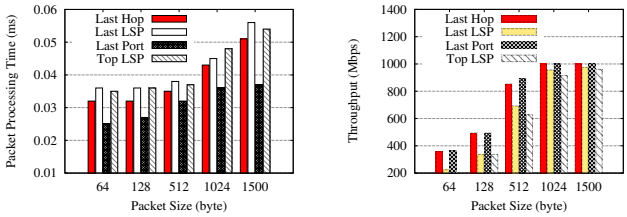


Fig. 6. An example of a rerouting LSP constructed by root switch S_1 to reroute packet sent by host H_1 to H_{16} via the source edge switch during a link failure between S_1 and S_{11}.



(a) Packet Processing Time (b) Average Throughput

Fig. 7. LESS packet processing time and average throughput.

LSP to the source edge switch by using its LMAC address and the primary source LMAC address. Then, the rerouting LSP from the source edge switch to the destination is determined by using the source and destination alternative LMAC addresses. Finally, the two LSPs are combined as the final rerouting LSP label. For example, in Fig. 6, the link between S_1 and S_{11} fails. Switch S_1 first derives an LSP to reach the source edge switch S_{13}. Comparing the primary source LMAC "`1:1:1:3:1:3`" with its own location label "`1`", the common prefix is "`1`". After the prefix is removed from both LMAC address, the primary source LMAC address becomes "`1:1:3:1:3`". Then, the first and the last port number of the LMAC are removed ("`1:3:1`"). The first byte is deleted because S_{11} is a root switch only having downward ports. The last byte/port is discarded because the source edge switch should not forward the deviated packet to the host. Next, the switch derives the rerouting LSP from the source edge switch to the destination, "`2.3.1.1.4.2.4.2.4`". Combining the two LSPs, the final reroute LSP is "`3.2.1.4.4.4`" after selecting the even position numbers.

It is important to point out that once the rerouted packet arrives at the destination, the destination host will use the alternative LMAC addresses to direct the return traffic even though it might not notice this failure. Furthermore, LESS is able to tolerate multiple link failures. In this case, multiple alternative LMAC addresses should be carried in packet headers, and multiple rerouting LSP operations are performed at intermediate switches.

VI. Performance Evaluation

This section describes the evaluation of LESS's performance using a FPGA-based implementation and Click-based simulation. The LESS-enable switch is used to measure the overhead of LSP label processing. The simulation is intended for measuring end-to-end forwarding latency and the latency of rerouted packets.

A. LESS Packet Processing Time

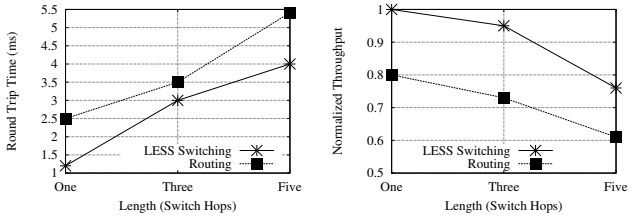
We develop a LESS switch prototype based on an Altera DE4 FPGA board (EP4SGX230KF40C2), which contains 4 Gigabit Ethernet ports, a PCI card, SRAM and DDR2 DRAM [18]. The prototype consists of two main components: LSP parser and packet rewriter. The LSP parser is used to derive the output port from an LSP label. The port number is then checked to see if the port is available. If it is, the LSP label header manipulation actions are performed at the packet rewriter. If the output port is not available, the packet is transmitted to the host processor to derive a rerouting LSP.

In order to measure LSP label processing time, we implement a traffic generator to generate LESS traffic toward the ports of the switch. By generating traffic with four different types of LSP labels, we measure the packet processing time and the corresponding throughput. We use 5 different frame sizes for the measurement, and for each frame size we repeat the measurement over 100 runs. Fig. 7(a) shows the latency for particular packet sizes. It can be observed that the type of LSP label headers has an impact on packet latency because of the overhead of LSP label manipulation. The third type of LSP label header has the least latency as expected.

The throughput is measured by using the maximum rate when the first loss occurs at the switch. Fig. 7(b) shows the throughput obtained for the LESS switch. The y-axis shows the amount of received frames in bytes per second. The figure shows that the LESS switch is capable of forwarding packets at a line-rate of 1Gbit/s.

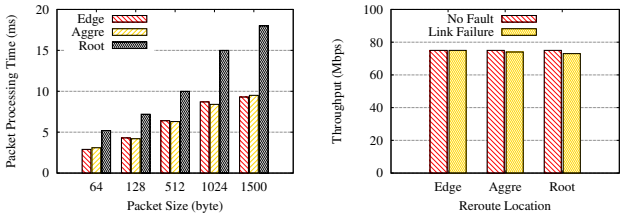
B. End-to-end Forwarding Latency

In order to measure the end-to-end forwarding latency, we built a testbed consisting of Dell PowerEdge servers with Intel Xeon CPUs running at 2.493 GHz clock speed. We deploy a fat-tree topology with 20 switches and 16 hosts in the testbed. We use the Click modular router [19] to implement software LESS switches and LESS agents. We compare the forwarding latency of LESS with a routing based approach, such as VL2 [17]. The routing approach measurement is considered as the baseline scenario. We measure the end-to-end forwarding latency in term of round-trip-time (RTT). We run iperf server and client at the hosts over 100 runs, and measure latency and throughput among those hosts. The evaluation results are shown in Fig. 8. We normalize the aggregate throughput by dividing by the link bandwidth of the network (100Mpbs). Fig.8(a) shows the average latency over different end-to-end path lengths. Fig.8(b) shows the average throughput of LESS and routing method. The results show that LESS outperforms the routing method in both delay and throughput. Meanwhile, the two figures show that the throughput and latency show a small degradation when the end-to-end path length is increased.



(a) Round Trip Time (b) Normalized throughput

Fig. 8. Round Trip Time and normalized throughput versus the end-to-end path length.



(a) Rerouting LSP Construction Time (b) Throughput during rerouting

Fig. 9. Latency of rerouting LSP construction and throughput during rerouting versus rerouting LSP construction location.

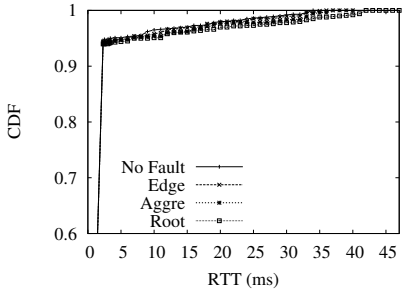


Fig. 10. A CDF of RTT when rerouting LSP construction performed by different switches.

C. Rerouting Latency

To evaluate the fault tolerance of LESS, we first measure the overhead of constructing a new LSP when a link failure is encountered in the FPGA-based switch. We manually configured the switch at different layers in a fat-tree topology. We then assign a set of LMAC addresses to it. The packet generator injects packets with a failed port to the switch. Based on its location, the switch performs different steps to construct the rerouting LSP. We consider the delay between the time when a packet is injected into the switch and the time when the packet with a new LSP is sent out as the latency. Fig. 9(a) shows the latency at different layers. We find that the worst performance is at the root switches because they need to calculate a backtracked LSP label. Comparing the worst latency (about 16 ms) of LESS with the convergence time of routing approach during a link failure (about 65 ms [14]), we find LESS can provide fast failover.

Next, we measure the throughput during a rerouting process based on the Click-based simulation. We use iperf to generate TCP flows with MTU of 512 bytes, and measure the latency and throughput over 100 runs. The average throughput during the rerouting process is shown in Fig. 9(b). We can see that the throughput during rerouting does not degrade significantly. In other words, rerouting LSP construction has limited impact on the throughput. Finally, we measure the RTT delay during rerouting. As seen in Fig. 10, the average latency is about 3 ms and the worst-case latency is 47 ms when the root switches detect a link failure. The root switch experiences the worst latency due to calculating and installing the new LSP label and extra propagation delay. Meanwhile, we find at most 244 packets are rerouted by using rerouting LSPs. The small number of rerouted packets is because that once the rerouted in-flight packets arrive at the source or the destination, the alternative LMAC addresses are used for subsequent packets to bypass the failure. Furthermore, the number of packets discarded because of each link failure is about 2 - 5 packets. LESS loses only the packets that are directly affected by a link failure, for example, the packets that are in transit over a failed link or queued in the network.

VII. Related Work

To achieve high performance of datacenter networks, most of the proposed datacenter structures embed topological information into IP addresses, or location-specific IP addresses to describe servers and switches' positions in the datacenter networks [14], [15], [17], [20]–[22]. For example, in PortLand [14], each node address is dependent on the node location. PortLand switches forward packets according to location-specific Pseudo-MAC address of the destination. ALIAS [23] is a decentralized and scalable protocol to assign host labels for the hierarchical topologies. Torii [24] is an address configuration protocol that can assign multiple hierarchical MAC addresses. Virtual Ethernet MAC addresses (shadow MACs) are proposed in work [12]. The key difference with the pseudo MAC address proposed in PortLand [14] and shadow MACs [12] is that LMACs are assigned based on the hierarchical structure of a multi-rooted tree.

Similar to our approach, many previous works employ source routing or label-based forwarding mechanisms in datacenter networks [3], [13], [25]. CONGA [26] uses source routing at the first hop to allow the destination to keep track of the path used. Work [11] shows that combining source routing and OpenFlow can provide a flexible architecture for datacenter networks. Path Switching [13] encodes interface labels into variable length strings. Work [25] proposes a fault-tolerance algorithm in fat-trees. However, the algorithm is not applicable beyond the fat-tree topology. In work [12], shadow MACs are used as forwarding labels in SDN networks. Axons [27] provides flexibility by supporting different route selection strategies. However, unlike our approach, the existing approaches require all source nodes to keep full topology information to compute the correct routes to a destination, which may not scale for large networks. For example, the

approach proposed in work [3] has potential scalability issues due to massive probing packets.

VIII. Conclusion and Future Work

The main contribution of the work is to leverage source switching as an alternative routing approach in datacenter networks. To support this new source switching architecture, we present a location-based addressing scheme. Based on the proposed LMAC addresses, each host and switch can locally derive a source route for an intended destination without requiring the full knowledge of the network topology. LESS simplifies switch design and does not require routing table lookup. Furthermore, we propose a dynamical fault tolerance method to avoid link failures. We illustrate the effectiveness of the source switching approach. Our evaluation results show that LESS can bring significant improvements in performance and reliability for datacenter networks.

The main overhead of LESS is the increased packet overhead due to the carried alternative LMACs. However, alternative LMAC addresses are not necessary to be carried in every packet header. LESS allows a host to determine when alternative LMAC addresses should be added into packet headers. The basic idea is that IP packets with alternative LMAC addresses are sent only when there is a failure predication made by the host, or performance bottlenecks are detected by the host. For example, the host can utilize TCP zero window probing and duplicate ACKs to predict a possible failure. In our future work, we will investigate different strategies and study their effectiveness on the reliability of datacenter networks.

Acknowledgment

This work was supported by National Science Foundation grant CNS-1402857 and CNS-1402594 and under NSF-NICT Collaborative Research JUNO (Japan-U.S. Network Opportunity) Program.

References

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," NSDI'10, pp. 19–19, 2010.

[2] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," CoNEXT '11, pp. 8:1–8:12, 2011.

[3] X. Jin, N. Farrington, and J. Rexford, "Your Data Center Switch is Trying Too Hard," in *ACM Symposium on SDN Research (SOSR 2016)*, (Santa Clara, California), March 2016.

[4] W. Braun and M. Menth, "Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking," in *Proceedings of the 2014 Third European Workshop on Software Defined Networks*, EWSDN '14, pp. 25–30, 2014.

[5] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A Software Defined Internet Exchange," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 551–562, August 2014.

[6] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An Industrial-scale Software Defined Internet Exchange Point," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pp. 1–14, 2016.

[7] M. Walraed-Sullivan, A. Vahdat, and K. Marzullo, "Aspen Trees: Balancing Data Center Fault Tolerance, Scalability and Cost," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pp. 85–96, 2013.

[8] K. Levchenko, G. M. Voelker, R. Paturi, and S. Savage, "Xl: An Efficient Network Routing Algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 15–26, August 2008.

[9] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring Connectivity via Data Plane Mechanisms," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pp. 113–126, 2013.

[10] A. Hari, U. Niesen, and G. Wilfong in *2015 IEEE International Symposium on Information Theory (ISIT)*.

[11] S. A. Jyothi, M. Dong, and P. B. Godfrey, "Towards a Flexible Data Center Fabric with Source Routing," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, (New York, NY, USA), pp. 10:1–10:8, 2015.

[12] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow MACs: Scalable Label-switching for Commodity Ethernet," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pp. 157–162, 2014.

[13] A. Hari, T. V. Lakshman, and G. Wilfong, "Path Switching: Reduced-State Flow Handling in SDN Using Path Information," in *CoNEXT 15*, (Heidelberg, Germany), December 2015.

[14] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric," SIGCOMM '09, pp. 39–50, 2009.

[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," SIGCOMM '08, pp. 63–74, 2008.

[16] Cisco, "Cisco Data Center Infrastructure 2.5 Design Guide." http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5_book.html, 2007.

[17] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," SIGCOMM '09, pp. 51–62, 2009.

[18] Altera, "Altera DE4 Development and Education Board." http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=501.

[19] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Trans. Comput. Syst.*, vol. 18, pp. 263–297, August 2000.

[20] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers," SIGCOMM '08, pp. 75–86, 2008.

[21] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, "DAC: Generic and Automatic Address Configuration for Data Center Networks," *IEEE/ACM Trans. Netw.*, vol. 20, pp. 84–99, February 2012.

[22] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," SIGCOMM '09, pp. 63–74, 2009.

[23] M. Walraed-Sullivan, R. N. Mysore, M. Tewari, Y. Zhang, K. Marzullo, and A. Vahdat, "ALIAS: Scalable, Decentralized Label Assignment for Data Centers," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pp. 6:1–6:14, 2011.

[24] E. Rojas, G. Ibanez, J. M. Gimenez-Guzman, D. Rivera, and A. Azcorra, "Torii: Multipath Distributed Ethernet Fabric Protocol for Data Centres with Zero-loss Path Repair," *Trans. Emerg. Telecommun. Technol.*, vol. 26, pp. 179–194, February 2015.

[25] F. O. Sem-Jacobsen, T. Skeie, O. Lysne, and J. Duato, "Dynamic Fault Tolerance in Fat Trees," *IEEE Transactions on Computers*, vol. 60, no. 4, pp. 508–525, 2011.

[26] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed Congestion-aware Load Balancing for Datacenters," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 503–514, August 2014.

[27] J. Shafer, B. Stephens, M. Foss, S. Rixner, and A. L. Cox, "Axon: A Flexible Substrate for Source-routed Ethernet," in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '10, pp. 22:1–22:11, 2010.