XOR-based Source Routing

Jérôme Lacan, Emmanuel Lochin

XOR-based Source Routing

Jérôme Lacan[1] and Emmanuel Lochin[2]
[1] ISAE-SUPAERO, Université de Toulouse, Toulouse, France
[2] ENAC, Toulouse, France
jerome.lacan@isae-supaero.fr, emmanuel.lochin@enac.fr

***Abstract*—We introduce a XOR-based source routing (XSR) scheme as a novel approach to enable fast forwarding and low-latency communications. XSR uses linear encoding operation to both 1) build the path labels of unicast and multicast data transfers; 2) perform fast computational efficient routing decisions compared to standard table lookup procedure without any packet modification all along the path. XSR specifically focuses on decreasing the complexity of forwarding router operations. This allows packet switches (e.g, link-layer switch or router) to perform only simple linear operations over a binary vector label which embeds the path. XSR provides the building blocks to speed up the forwarding plane and can be applied to different data planes such as MPLS or IPv6. Compared to recent approaches based on modular arithmetic, XSR computes the smallest label possible and presents strong scalable properties allowing to be deployed over any kind of core vendor or datacenter networks. At last but not least, the same computed label can be used interchangeably to cross the path forward or reverse in the context of unicast communication.**

I. Introduction

Source routing is a very old technique to route a data packet from a source to a destination, initially presented in [1] and currently developed at the IETF within the SPRING (Source Packet Routing in Networking) working group [2]. Compared to conventional routing that forwards packets following both the IP destination address and the forwarding table lookup, source routing allows the sender to partly or completely indicate inside the packet headers the path that must be followed. Source routing brings out several advantages. As highlighted in [3], the data plane becomes simpler, core elements perform simple operations and traffic engineering is more flexible.

Source routing technique gained in popularity in particular following the rapid spread of Software Defined Networking (SDN) paradigm as a scalable solution to deploy services in datacenters [4]. In particular, the authors in [5] illustrate that SDN-based source routing significantly decrease flow-states exchange by storing the path information into packet headers. Encoding the whole path inside a packet suppresses expensive lookup procedures inside core packet switches (e.g, link-layer switch or router) as each switch is able to quickly identify the next hop of the path stored in the packet.

The length of the encoded path label is one of the potential issues in source routing. In particular, there are use cases where each individual hop must be specified in the label resulting in a long list of hops that is instantiated into a MPLS label stack (in the MPLS data plane) or list of IPv6 addresses (in the IPv6 data plane) [2]. Obviously, this leads to potentially oversized labels. Furthermore, current MPLS equipments only support limited number of stacked labels (five to ten labels are currently supported by some routers [6]). To cope with this problem, there exists an up to date variant called segment routing [7] that leverages source routing principle. Segment Routing encodes a path label as a stack composed by node segments (a router) and adjacency segments (a router interface output) [7] which prevents to record all nodes addresses.

XOR-based source routing (XSR) scheme is a novel approach to improve data plane operations enabling fast forwarding. The originality of XSR is to conjointly optimize the size of the path label with low switching processing cost while enabling multicast and unicast forwarding. This is explained by the use of linear operations over binary vectors. A large survey browsing previous attempts is proposed in [8]; eight papers are identified therein. In this study, we choose to select and focus on two recent competitive works that provide path optimization techniques to minimize the size of a path label encoded inside packets. We will mainly discuss XSR against these two solutions proposed respectively in 2017 [9] and in 2018 [10]. In the latter, the authors propose a whole architecture, referred to as RDNA, that lays on modular arithmetic to compute a label number to identify (following a reverse operation) the output switch port considering a unique router ID [10].

After presenting our proposal, we will show in VI that RDNA requires larger path label length and performs less computational efficient operations than XSR (i.e. XOR versus modulo operation), in particular for multicast. In [9], the authors propose an elegant algorithm that minimizes the maximum length of any encoded path in the network. The main drawback is the restriction of this solution to unicast exchanges. On the contrary, XSR copes with all these issues enabling unicast and multicast communications at the same processing cost, performing fast and low latency routing decisions without any packet modification.

II. The Path Encoding Problem

Actually, a router only needs to assess the output link(s) corresponding to a given input packet. So, the path of a packet can be encoded by the output links sequence of the routers composing the path. Since the labels of the output links (denoted interface labels in the following) are local to a node, they can be represented by short bit vectors. For example, a node having 3 output paths can number them 0, 1 and 2 and thus, uses 2-bit vectors (00), (01) and (10) as interface

labels to identify them. This principle, adopted by the authors in [11], uses short fixed-length interface labels. Note that the number of bits needed to identify each interface label of a router depends on the number of output links. The authors in [9] further investigate this approach by using variable-length prefix-codes usually used in lossless compression systems to represent the interface labels of the output links. They show that they can reduce the lengths of the largest encoded paths.

With segment routing, all the labels have the same length and each router considers the first label at the top of the stack in the received packet, processes the packet, then removes this label. The next router uses the next label until the final receiver. When short interface labels are used, this strategy can not be applied because the interface labels size is not necessarily a multiple of 8 bits. In [11], each interface label has a fixed length and a hop counter is added to the header allowing to identify the current path position. This counter is then decremented by each router before forwarding the packet involving data modifications. Similarly to [11], due to variable interface label sizes, a pointer is also needed in [9] to point the current position in the encoded path. After reading its corresponding interface label, the router slides the pointer and forwards the packet.

The localization strategies of the labels in the encoded paths have several implications. First, removing or modifying some parts of the label involves header supplementary data operations and computations. The second consequence is that these strategies are only usable for unicast transmissions. Indeed, if we consider multicast or multipath transmissions, some router must send some packets on several interfaces. However, the header modifications done by the router are only based on its local information and thus it is not possible to make different modifications on the packets sent to the different interfaces.

Other strategies have been proposed to enable multipath or multicast. In [12], the source builds a Bloom filter which is based on the addresses of the nodes of the path and which is stored in the packet header. At the reception of a packet, each router checks whether the addresses of its neighbours are verified by the Bloom filter and forwards the packet to the valid ones. Since Bloom filters are probabilistic tools, the main difficulty with this scheme is to choose the right parameters of the filter to minimize the ratio of false positive while maintaining a reasonable size. Finally for multicast transmissions, where a data can be sent to different interfaces, the interface label can be chosen as a bitmap. For example, the label of a packet that must be sent on the interfaces $0, 4$ and 5 of a router having 6 interfaces is (110001). A simple method to generate the encoded path is to concatenate the interface labels. More elaborated strategies presented in [13] and [10] encode the path into an integer number. The routers recover their information by computing the residue of this integer modulo a prime number.

III. XOR-based Source Routing in a Nutshell

Let us start with an illustration of XSR interface labels principle. We recall that an interface label corresponds to an interface IDs of a router or a set of interfaces in case of multicast. Fig. 1 shows an example where an input packet of a unicast transmission coming from the interface $(4 = 100_b)$ is forwarded to the interface $(3 = 011_b)$. The interface label of this packet for this router is defined as the XOR between the input interface ID and the output interface ID, *i.e.* $100_b \oplus 011_b = 111_b$. It is obvious that the output interface ID can be computed from the interface label and the input interface ID ($100_b \oplus 111_b = 011_b$). Similarly, for the reverse path, the input interface ID can be retrieved from the output interface ID and the interface label ($011_b \oplus 111_b = 100_b$).

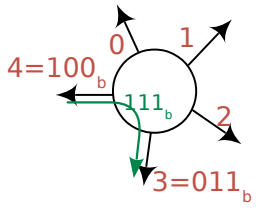

Fig. 1. Unicast interface label

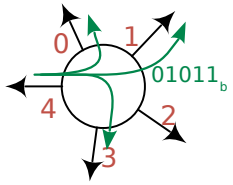

Fig. 2. Multicast interface label

For a multicast transmission Fig. 2, the interface label to the packet is the bitmap (01011) of the output interface IDs (to be read from the right to the left). A 1 in positions 0, 1 and 3 means the packet must be forwarded to the ports 0, 1 and 3.

XSR principle is to concatenate the interface labels of each router of the path into a vector L. We assume that each router has a unique identifier R_{ID} (e.g. hardware address). The path label, denoted P, is computed by the source (or directly provided by a centralized SDN controller) by applying to L a linear transformation based on the IDs of the routers of the path. This path label is stored in the header of the packet. To forward a packet, each router applies a filtering function (based on its own ID) to the path label to get its interface label. This function is a linear function over the binary finite field $\mathbb{F}_2$ only using XOR-based operations.

The first advantage is that packets are not modified when crossing a router. On the contrary to [9], the interface labels list does not need to be ordered in the path label preventing the use of pointer or vector. This filtering function is simply few dot products of short vectors that can be done on-the-fly, compliant with fast routing strategies like e.g. cut-through.

In brief, the length of the path label P is the sum of the lengths of the interface labels of each router of the path, even if they do not have the same length.

To illustrate this, let us consider the network presented in Fig. 3. Assume that the source S requests to send a packet to the destination D through the path $(R_{17}|R_{11}|R_{29})$. The path can be represented by the sequence of interface labels $L = (L_{17}\ L_{11}\ L_{29}) = (2 \oplus 0\ 0 \oplus 1\ 1 \oplus 2) = (10_b \oplus 00_b\ 0_b \oplus 1_b\ 01_b \oplus 10_b) = (10_b\ 1_b\ 11_b)$. Once again, the lengths of

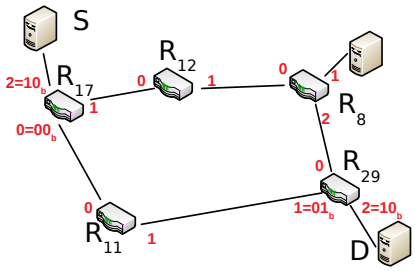

Fig. 3. Example network

the interface labels varies according to the routers or can be variable for the same router like in [9]. Here, we consider that R_{17} and R_{29} labels have a length of 2 bits while R_{11} label has a length of 1 bit.

The path label P is computed by solving the following system built from the filtering functions F_{17}, F_{11} and F_{29}:

$$F_{17}(P) = (10) \quad (1)$$
$$F_{11}(P) = (1) \quad (2)$$
$$F_{29}(P) = (11) \quad (3)$$

These functions are linear operations characterized by a matrix defined from the routers IDs. For R_{17}, let us denote this matrix M_{17}. We then have:

$$F_{17}(P) = P \cdot M_{17} = (10)$$

where the notation $\cdot$ between two vectors or matrices represents the matrix multiplication. Since the length of P is the sum of the lengths of the interface labels, i.e. 5, and the length of the interface label is 2, M_{17} has 5 rows and 2 columns. In this simple example, the first column is defined by the router ID $17 = 010001_b$ and the second column as its cyclic shift. Finally the label must verify:

$$F_{17}(P) = P \cdot M_{17} = P.\begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} = (10) \quad (4)$$

By using the same method, we can obtain the following linear equations for R_{11} and R_{29}:

$$P \cdot M_{11} = P \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = (1) \text{ and } P \cdot M_{29} = P \cdot \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} = (11) \quad (5)$$

The set of linear combinations can be aggregated in the 5×5 matrix $M = (M_{17}|M_{11}|M_{29})$ allowing to obtain the global relationship between the path label P and the concatenation of interface labels list L:

$$P \cdot M = L \quad (6)$$

By observing that M is invertible, the source computes M^{-1} and P as follows:

$$P = L \cdot M^{-1} = (10\ 1\ 11) \cdot \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix} = (11100) \quad (7)$$

It can be verified that (1), (2) and (3) hold for this value of P.

IV. XOR-based Source Routing

We have previously illustrated within a little example the main principle of our proposal. We now present in further details the core mechanisms of XOR-based source routing.

A. Network Hypotheses

We define a network has a set of edge nodes (source and/or destination nodes) connected to n_R routers R_j, $j = 1, \ldots, n_R$ as illustrated Fig. 3. Communications occur between several edge nodes through a path formed by several routers. For example, a unicast communication between a source S and destination D could use either the path $(R_{17}|R_{50}|R_{29})$ or $(R_{17}|R_{12}|R_{39}|R_{29})$. The connection can be unicast, multipath or multicast.

B. Router Forwarding

The main operation done by a router R_i at the reception of a packet is to filter the path label P to recover its corresponding interface label L_i.

The general form of the simple example presented in III is to consider that each router stores 2^ϵ binary filtering matrices $\overline{M}_i^{(e)}$, for $e = 0, \ldots, 2^\epsilon - 1$. Each matrix has $\overline{s_P}$ rows and $\overline{s_i}$ columns, where $\overline{s_P}$ is the maximum length in bits of the size of P and $\overline{s_i}$ is the maximum size of the interface labels of R_i.

At the reception of a packet, the router reads the path label P of size s_P and the value e set by the source stored on ϵ bits (further explained in the following).

Since $s_P \leq \overline{s_P}$ and $s_i \leq \overline{s_i}$, the router takes M_i as the submatrix of $\overline{M}_i^{(e)}$ formed by the first s_l rows and s_i columns.

The filtering function F_i is a set of linear operations which consists in multiplying the path label P by a filtering matrix M_i with s_i columns and s_P rows.

More formally, if $\mathbb{F}_2$ denotes the binary finite field, the filtering function is defined as follows:

$$F_i \begin{cases} \mathbb{F}_2^{s_P} & \longrightarrow & \mathbb{F}_2^{s_i} \\ P & \longrightarrow & P \cdot M_i \end{cases}$$

These operations are summarized in Fig. 4

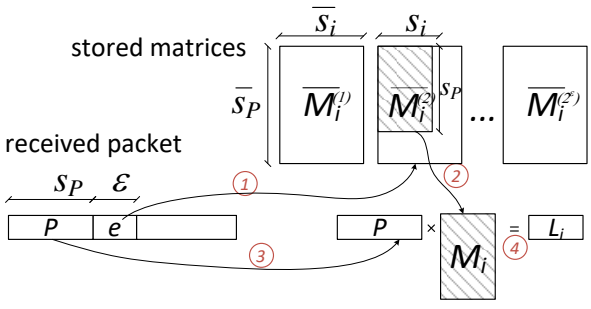

Fig. 4. Receiver operations. The numbers represent the different steps.

C. Path Label Construction

Once again, the construction of a path label from a source S to a destination D is done either by the source itself or by the controller which builds the path label and send it to the source.

Let $\{R_{i_1},\ R_{i_2}, \ldots, R_{i_p}\}$ the set of the routers on the path. For unicast transmission, this set corresponds to a sequence of routers. Considering multicast, this sequence is not ordered and corresponds to the set of routers that will forward the packet.

The concatenation of the interface labels corresponds to a bit vector denoted $L = (L_{i_1},\ L_{i_2}, \ldots, L_{i_p})$ with a size $s_L = \sum_{k=1}^{p} s_{i_k}$.

As seen in the previous paragraph, the routers multiply the path label by their filtering matrix to obtain their interface label. This implies that the path label P must verify some linear equations. These equations can be represented by the matrix $M = (M_{i_1},\ M_{i_2}, \ldots, M_{i_p})$ which is the concatenation of the filtering matrices used by the routers of the path. Since we set s_P the length of P, to s_L, M is a $s_L \times s_L$-square matrix. We define a path label as *valid* if the filtering process (defined in the previous section) applied by any router of the path produces the correct interface label of a given router. We will show that we can obtain a valid path label P if M is nonsingular.

The construction of P is based on the following theorem:

Theorem 1. *Let M^{-1} be the inverse of M. Then:*

$$P \stackrel{\text{def}}{=} L \cdot M^{-1} \tag{8}$$

is a valid path label.

Proof. From the definition of P, we have $P \cdot M = L \cdot M^{-1} \cdot M = L = (L_{i_1},\ L_{i_2}, \ldots, L_{i_p})$. On the other hand, $P \cdot M = P \cdot (M_{i_1},\ M_{i_2}, \ldots, M_{i_p}) = (P \cdot M_{i_1},\ P \cdot M_{i_2}, \ldots, P \cdot M_{i_p})$. It follows that, for each $k = 1, \ldots, p$, $P \cdot M_{i_k} = L_{i_k}$ and thus P is valid. □

Theorem 2. *Let P be a valid path built from Theorem 1 to route the packets for a unicast transmission from a sender to a destination.*

Then, P is also valid to route the packets from the destination to the source.

Proof. To prove this theorem, it is sufficient to prove that if a router receives a packet with a path label P on an input interface ID_i and forwards it to the output interface ID_o, then, if it receives a packet with the same path label on the interface ID_o, it forwards it to the interface ID_i.

Let us consider a router R_u of the path. On the forward path, the router filters the path label of a packet with the matrix M_u and obtains the interface label $L_u = P.M_u$. According to III, L_u is the XOR of the input interface ID_i and the output interface ID_o. Since it knows ID_i, it is able to recover the ID_o by XORing L_u and ID_i ($ID_i \oplus L_u = ID_i \oplus (ID_i \oplus ID_o) = ID_o$). Then it transmits the packet on the output interface.

On the reverse path, it receives a packet from the interface ID_o with the same path label. By applying the filtering function M_u, it recovers L_u. Since it knows the packet arrived from interface ID_o, it XORs ID_o and L_u and obtains ID_i ($ID_o \oplus L_u = ID_o \oplus (ID_i \oplus ID_o) = ID_i$). Then it forwards the packet to the interface ID_i. □

V. Analysis of the Parameters

The main system parameter that must be evaluated is the probability of building a valid path label. Indeed, this probability impacts on the complexity of the construction of a path label and allows to correctly size ϵ previously presented in IV-B. Note we make no assumption on the filtering matrices and consider them as random binary matrices.

A. Probability of Construction of a Valid Path Label

According to IV-C, a valid path label can be built if the matrix M is invertible. [14] shows that the probability that a $s_L \times s_L$ binary random matrix is invertible is equal to

$$\sigma_0(s_L) = \sum_{i=0}^{s_L - 1} (1 - 2^{s_L - i}) \tag{9}$$

Fig. 5 confirms that this probability quickly converges to the limit which is known to be 0.2888.

A valid path can be build if at least one of the 2^ϵ matrices built from the matrices stored by the routers is invertible. The probability is thus equal to

$$\sigma_\epsilon(s_L) = 1 - (1 - \sigma_0(s_L))^{2^\epsilon} \tag{10}$$

These values are plotted Fig. 6. It can be observed that a valid path can be obtained with a very high probability (for example, 0.99998 for $2^5 = 32$ 8×8 binary matrices stored by each router).

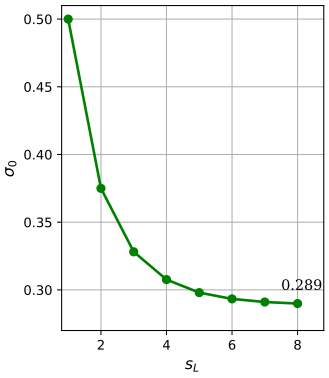

Fig. 5. proba. invertible matrix

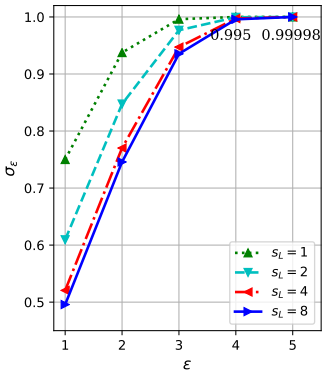

Fig. 6. proba. valid path

B. Complexity of the Path Label Construction

As the path label computation can also be done in the control plane (e.g. SDN controller), there is no impact on the data plane forwarding procedure. However, we believe that estimating its complexity is of interest.

To build a path label from a matrix M, it is necessary to find its rank and to perform a matrix inversion. This is generally

done by using algorithms based on Gaussian Elimination (GE). Even if there exists theoretical optimizations like Strassen's algorithm [15] which runs in $\mathcal{O}(n^{2.807})$ operations for very large matrices, we will consider that the complexity is in $\mathcal{O}(n^3)$ for each tested matrix.

To evaluate the average number of tested matrices, we can observe that it is necessary to perform k GEs only when the first $k-1$ fail to build a valid path and when the k^{th} succeeds. This occurs with the probability $(1-\sigma)^{k-1}\sigma$. It follows that the average number of GEs is:

$$\sum_{k=1}^{\infty} k(1-\sigma)^{k-1}\sigma = \frac{1}{\sigma} \tag{11}$$

A value of $\sigma = 0.6103$ gives an average number of GEs equal to $1/0.6103 = 1.6385$.

C. Number of Signalling Bits

Section V-A has shown that σ must be chosen greater or equal to 3 to provide a high probability of building a path label. This represents the size of the signalling field added to the packet header with the path label. For example, this value is similar to the size of the pointer used in [9] which is 4 bits in most of studied configurations.

D. Storage Amount in Routers

According to IV-B, each router stores 2^{ϵ} binary matrices of $\overline{s_P}$ rows and $\overline{s_i}$ columns. So the global amount stored by a router is

$$2^{\epsilon} \times \overline{s_P} \times \overline{s_i}$$

By considering unicast transmissions, reasonable maximal values of $\epsilon = 4$, $\overline{s_P} = 50$ and $\overline{s_i} = 10$ can be chosen. This represents $50 \times 10 \times 16 = 8000$ bits, *i.e* 1000 bytes which is completely scalable.

For multicast transmissions, the path label can be larger (see VI-B). In the largest studied case, the path label has a size of around 200 bytes and the interface labels have a maximal size of around 100 bits. For $\epsilon = 4$, the total number of bits stored is $16 \times 200 \times 8 \times 100 = 2.56$ Mbits *i.e* 320 Kbytes. Even if this number is rather low compared to traditional routers, we can easily reduce it by globally optimizing the choice of the filtering matrices in order to reduce the value of ϵ. Another possibility is to use filtering matrices that can be deduced from a short representation as in III where the columns of the filtering matrices are deduced from the first column by cyclic permutations.

VI. XSR Versus Existing Work

Two recent results have interesting relationships with our proposal. In the two next sections, we expose these links and compare various metrics of interest.

A. Optimal Path Encoding for Unicast Transmissions

This first considered work, denoted OPE, is presented in [9]. The authors propose to use prefix codes to represent the interface labels and optimize the choice of the labels in order to minimize the maximal length of the path label. The path label is then the sequence of the interface labels with an additional pointer indicating to a router the position in the encoded path that it must consider. This pointer is updated by router according to the length of its interface label. This scheme allows to reduce significantly the size of the largest path label.

Compared to this work, our proposal goes further by encoding their output (the sequence of optimized interface labels) with binary linear operations.

If we estimate the amount of bits needed to implement each solution, the lengths of the path labels are equal both for OPE and XSR and require the same amount of signalling bits: around 4 for OPE to encode the pointer and $\epsilon = 3$ or 4 with XSR.

However, the advantage to add XSR on top of OPE is twofold:

1) the pointer used by OPE involves ordered sequence of interface labels and thus can only be used for unicast transmissions. This is rather unfortunate because the idea of optimizing the interface labels according to the maximal length makes sense for multicast transmissions as in datacenter networks (see next section). Encoding the path with XSR removes this notion of order and thus allows multicast transmissions;
2) using fast filtering router operations allows to prevent any packet modification due to pointer update or possibly integrity checks.

B. Datacenter Networks

1) Recent Work in Source Routing for Datacenter Networks: the potential of source routing for datacenter networking was demonstrated in KeyFlow [16] and COXcast [13] for both unicast and multicast transmissions. The first interest is the simplification of the management of multiple small multicast groups. The protocol Xcast [2] was defined for this purpose. However, the generated headers can be large. To cope with this issue, KeyFlow and COXcast independently propose a source routing mechanism encoding the paths with interface labels associated to the interfaces of the routers. The main idea is to associate to each router a prime number label and to the paths an integer stored in the packet header. At the reception of a packet, a router simply computes the residue of the path modulo its label. The obtained value corresponds to the output interface(s). They reduce significantly the size of the path label compared to Xcast. Moreover, the core routers neither use forwarding tables nor modify the packets. This simplifies router operations and reduces the processing delay allowing ultra-low latency communications.

The path label size is also reduced in the RDNA architecture [10]. RDNA improves the way to choose the prime numbers and to compute the path. Since the integer path is determined

from the prime numbers of the system, it is preferable to use short prime numbers in order to reduce the size of the integer path. Unfortunately, the amount of primes in integer numbers is quite low and it is not always possible to choose small prime numbers that provide residues with a given number of bits. Multiplying prime numbers (and finding the right ones) leads to oversized binary values, making the path label size not optimal and RDNA solution less flexible than XSR in particular in the context of multicast.

2) Path Length Comparison: the mechanism used in COXcast and RDNA lays on a concept similar to XSR. The main difference is that XSR is based on linear algebra while both others are based on modular arithmetic. Linear algebra leads to several advantages:

- linear algebra does not have the problem of scarcity of prime numbers and thus the length of the path is very close to the optimal. This is demonstrated in Tables I and II;
- linear operations performed in routers are simple dot products and are less complex than modulo operations on integers;
- linear algebra provides a better flexibility. The configuration of the global network can easily be changed because finding new invertible matrices is effortless and leads to optimal size compared to the complex choice of the best set of prime numbers.

We now compare the overhead in terms of sizes. We consider the use cases studied in RDNA [10] and compute the corresponding header length for each solution.

The datacenter network analyzed in [10] is a 2-tier Clos network topology (shown Fig. 7) composed of two stages core switches (spine and leaf) and one stage of edge switches connected to hosts. The connections are defined between two hosts. The considered path is defined between the edge switches respectively connected to the hosts source and destination. The longest path is from the edge switch connected to source to the edge switch connected to the destination through a first leaf, a spine and a second leaf.

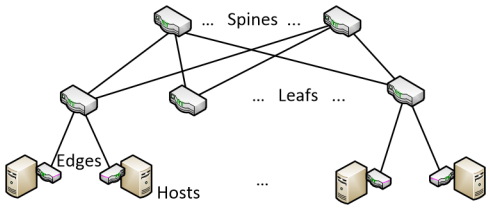

Fig. 7. Datacenter network

Let us denote `spines`, resp. `leafs`, the number of spines, resp. leafs, and let `ports` be the number of ports of the leafs. The number of ports of the spines is `leafs`.

To represent a unicast path, we then need to store the output port of the first leaf (i.e. among $\texttt{ports}-1$ since we do not consider the input port), then the output port of the spine (among $\texttt{leafs}-1$) and then the output port of the second leaf (among $\texttt{ports}-1$).

According to the results of IV-C, this path can be encoded in $2.\log_2(\texttt{ports}-1)+\log_2(\texttt{leafs}-1)+\epsilon$ bits with a high probability. We fix the value of ϵ to 4 bits[1].

The number of bytes necessary to encode the path is thus:

$$\left\lceil (2.\log_2(\texttt{ports}-1)+\log_2(\texttt{ports}-1)+4)/8 \right\rceil$$

The obtained values are compared to COXcast and RDNA in Table I. We observe that we always have path label sizes lower or equal to the other proposals.

For multicast transmissions, the longest path is from the host source and its corresponding edge switch to all other hosts. The packet must be sent from the corresponding edge switch to a first leaf which forwards it to all its ports connected to other edge switches and to one spine. The spine transmits the packets to all others leafs which forwards it to all their connected edge switches (see Fig. 4 of [10]).

We recall that multicast interface labels can be represented as a bitmap of the output ports. Thus, a multicast interface label of a spine is a vector of `leafs` bits and an interface label of a leaf is a vector of `ports` bits.

The application of results of IV-C leads to a encoded path of length `ports`+`leafs`+(`leafs`-1).`ports`+ϵ bits. By fixing the value of ϵ to 4 bits, we obtain the following number of bytes:

$$\left\lceil (\texttt{ports}+\texttt{leafs}+(\texttt{leafs}-1).\texttt{ports}+4)/8 \right\rceil$$

The values obtained are reported in Table II in the row "XSR v1". Except two cases, a small gain is observed in most configurations.

The rather intuitive representation of our filtering operation allows us to propose an enhancement of the path encoding in the multicast case. The idea is to optimize the interface label in the leafs by differentiating the ports of the leafs connected to the spines and the one connected to edge switches. We propose to use some "signalling" bits in the label to encode differently the packets that must be only sent to some spines, the ones that must be only sent to edge switches and the others. To reduce the number of these bits, we use the prefix code $\{0, 10, 11\}$ as it was proposed in [9] for unicast transmissions.

In a use case, for $\texttt{spines}=6$ and $\texttt{ports}=16$, the new labels would be:

- [10.....]: the code 10 followed by the 6 ports connected to the spines for the packets only sent to some spines
- [0.........]: the code 0 followed by the $16-6=10$ ports connected to the edges for the packets that only be sent to the edges.
- [11.................]: the code 11 followed by the 16 ports for the others packets

This leads to an encoded path of length $2+\texttt{ports}+\texttt{leafs}+(\texttt{leafs}-1).(1+\texttt{ports}-\texttt{spines})+\epsilon$ bits.

[1]The value of ϵ can be reduced by determining a static configurations of the filters (out of the scope of this paper).

TABLE I
PATH LABEL SIZE (BYTES) FOR UNICAST

Spine	2			6					12					8				
Leafs	4			12					16					16				
Ports	16	24	32	16	24	32	48	96	16	24	32	48	96	16	24	32	48	96
COXcast[13]	5	8	11	5	8	11	17	35	5	8	11	17	35	5	8	11	17	35
RDNA [10]	2	2	2	3	3	3	4	4	3	3	3	4	4	3	3	3	4	4
XSR	2	2	2	2	3	3	3	3	2	3	3	3	3	2	3	3	3	3

TABLE II
PATH LABEL SIZE (BYTES) FOR MULTICAST

Spine	2			6					6					8				
Leafs	4			12					16					16				
Ports	16	24	32	16	24	32	48	96	16	24	32	48	96	16	24	32	48	96
COXcast[13]	10	14	18	36	48	60	84	156	47	63	79	111	207	51	67	83	115	211
RDNA [10]	9	14	18	26	39	52	75	154	34	51	68	100	200	34	51	68	100	200
XSR v1	9	13	17	26	38	50	74	146	35	51	67	99	195	35	51	67	99	195
XSR v2	9	13	17	20	32	44	68	140	26	42	58	90	186	22	38	54	86	182

By fixing the value of ϵ to 4, we obtain the following number of bytes:

$$\lceil (2 + \texttt{ports} + \texttt{leafs} + (\texttt{leafs} - 1). \\ (1 + \texttt{ports} - \texttt{spines}) + 4)/8 \rceil$$

From a practical point of view, to forward a packet with the types of labels, the leaf just needs to identify the 2 first bits to determine the length of the label it must recover and the filter it must use.

The results reported in the row "XSR v2" of Table II show a significant gain in most use cases.

VII. Conclusion and Future Work

We presented XOR-based source routing, a new data plane scheme enabling fast forwarding by performing only simple linear operations over a binary vector label which embeds an encoded routing path label. Compared to recent approaches, XSR computes the smallest label possible and does not need to modify forwarded packets. The main advantage compared to other existing approaches is to allow the re-use of the same path label for the feedback path and so, prevent the receiver to compute another label to reply (considering the SDN controller allows the same path for reply). XSR provides the building blocks to speed up the forwarding plane and can be applied to different data planes such as MPLS or IPv6 for unicast and multicast communications.

In a future work, we expect to implement XSR within Mininet emulator to further demonstrate the effective processing cost of forwarding operations.Furthermore, we believe that XSR would lead to promising application in terms of privacy and security if routers filtering operations remain unknown to attackers attempting to observe the network. Last but not least, we wish to present and discuss this solution at the IETF.

References

[1] "Internet Protocol," RFC 791, Sep. 1981. [Online]. Available: https://rfc-editor.org/rfc/rfc791.txt

[2] S. Previdi, C. Filsfils, B. Decraene, S. Litkowski, M. Horneffer, and R. Shakir, "Source Packet Routing in Networking (SPRING) Problem Statement and Requirements," RFC 7855, May 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7855.txt

[3] T. Lee, C. Pappas, C. Basescu, J. Han, T. Hoefler, and A. Perrig, "Source-based path selection: The data plane perspective," in *Conference on Future Internet Technologies (CFI)*. ACM, Jun. 2015.

[4] A. Abujoda, H. R. Kouchaksaraei, and P. Papadimitriou, "SDN-based source routing for scalable service chaining in datacenters," in *Wired/Wireless Internet Communications*. Springer International Publishing, 2016, pp. 66–77.

[5] S. Li *et al.*, "Improving SDN scalability with protocol-oblivious source routing: A system-level study," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 275–288, Mar. 2018.

[6] R. Guedrez, O. Dugeon, S. Lahoud, and G. Texier, "A new method for encoding mpls segment routing te paths," in *2017 8th International Conference on the Network of the Future (NOF)*, Nov. 2017, pp. 58–65.

[7] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, Jul. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8402.txt

[8] P. L. Ventre *et al.*, "Segment Routing: a Comprehensive Survey of Research Activities, Standardization Efforts and Implementation Results," 2019.

[9] A. Hari, U. Niesen, and G. Wilfong, "On the problem of optimal path encoding for software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 189–198, Feb. 2017.

[10] A. Liberato *et al.*, "RDNA: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1473–1487, Dec 2018.

[11] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Source routed forwarding with software defined control, considerations and implications," in *Proceedings of the ACM Conference on CoNEXT Student Workshop*. New York, NY, USA: ACM, 2012, pp. 43–44.

[12] C. Castelluccia and P. Mutaf, "Hash-based dynamic source routing," in *Networking 2004*, 2004, pp. 1012–1023.

[13] W. Jia, "A scalable multicast source routing architecture for data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, pp. 116–123, January 2014.

[14] E. R. Berlekamp, "The technology of error-correcting codes," *Proceedings of the IEEE*, vol. 68, no. 5, pp. 564–593, May 1980.

[15] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, no. 4, pp. 354–356, Aug. 1969.

[16] M. Martinello, M. R. N. Ribeiro, R. E. Z. de Oliveira, and R. de Angelis Vitoi, "Keyflow: a prototype for evolving sdn toward core network fabrics," *IEEE Network*, vol. 28, no. 2, pp. 12–19, March 2014.