Simplifying Forwarding Data Plane Operations with XOR-Based Source Routing

Jérôme Lacan, Emmanuel Lochin

Simplifying Forwarding Data Plane Operations With XOR-based Source Routing

Jérôme Lacan[1] and Emmanuel Lochin[2*]

[1]ISAE-SUPAERO; Université de Toulouse 4 avenue Édouard-Belin, Toulouse, 31400, Occitanie, France.
[2*]ENAC; Université de Toulouse 7 avenue Édouard-Belin, Toulouse, 31400, Occitanie, France.

*Corresponding author(s). E-mail(s): emmanuel.lochin@enac.fr;
Contributing authors: jerome.lacan@isae-supaero.fr;

Abstract

We propose a theoretical analysis of a novel source routing scheme called XSR. 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 computational complexity of forwarding operations. This allows packet switches (e.g, link-layer switch or router) to perform only simple linear operations over a binary vector label that embeds the path. We provide analytical proofs demonstrating that XSRs efficiently compute a valid unicast or multicast path label over any finite fields $\mathbb{F}_{\mathbf{2^w}}$. Furthermore, we show that this path label can be used for both the forward and return unicast paths, unlike other source routing algorithms that require recomputing a label for the return path. Compared to recent approaches based on modular arithmetic, XSR computes the smallest label possible and presents strong scalable properties, allowing it to be deployed over any kind of core vendor or datacenter networks.

Keywords: source/segment routing, optimal path label encoding, label switching, SDN

1 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, which 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 has several advantages. As highlighted in [3], the data plane becomes simpler, core elements perform simple operations, and traffic engineering is more flexible.

The source routing technique gained popularity in particular following the rapid spread of the Software Defined Networking (SDN) paradigm as a scalable solution to deploy services in datacenters [4]. Source routing is also the basis of embedded networks such as SpaceWire [5] or ad-hoc networks [6]. This is also the case for most wide-area networks managed by Internet Service Providers that use traffic engineering techniques to ease their network management. For instance, ISPs usually deploy centralized or distributed SDN controllers to handle their network, MPLS-TE to control routing paths or IPv6. Some studies have highlighted significant improvements in convergence times because of the reduced number of states that the SDN controller must distribute into the network.

In particular, the authors in [7] illustrate that SDN-based source routing significantly decreases 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 can quickly identify the next hop of the path stored in the packet. This explains why the translation of a network path resulting from a given TE objective to an SID sequence is a key operation for the deployment of segment routing [8].

The length of the encoded path label is a potential issue 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 an MPLS label stack (in the MPLS data plane) or a list of IPv6 addresses (in the IPv6 data plane) [2]. This obviously leads to potentially oversized labels. Furthermore, current MPLS equipments only supports a limited number of stacked labels (five to ten labels are currently supported by some routers [9]). To cope with this problem, there exists an up-to-date variant called segment routing [10] that leverages the source routing principle. Segment Routing encodes a path label as a stack composed of node segments (a router) and adjacency segments (a router interface output) [10] which prevents the recording of all nodes addresses. Another solution to reduce the path label size is to reduce the number of fixed-length labels of the path, as proposed in [11–13] or to use interface labels instead of global ones [14].

The XOR-based source routing (XSR) scheme is a novel approach to improve data plane operations.The originality of XSR is that it conjointly optimizes the size of the path label with low switching processing cost while enabling multicast and unicast forwarding. This is explained through 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 select and focus on two recent competitive studies

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 [15] and in 2018 [16]. 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 [16].

After presenting our proposal, we will show in 8 that RDNA requires a larger path label length and performs less computationally efficient operations than XSR (i.e., XOR versus modulo operation), particularly for multicast. In [15], 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, and performing computationally efficient routing decisions without any packet modification.

We first discuss the path encoding problem in the next Section 2. Then, we provide the big picture of our proposal through a simple illustrated example in Section 3. The full description of the XOR-based source routing is further detailed in Section 4. An analysis of XSR parameters is done in Section 5, followed by a comparison of XSR with existing approaches in Section 8. We conclude this work in Section 9.

2 The Path Encoding Problem

Actually, a router only needs to assess the output link(s) corresponding to a given input packet. Therefore, 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 [14], 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 [15] 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 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, and then removes this label. The next router uses the next label until it reaches the final receiver. When short interface labels are used, this strategy cannot be applied because the interface label size is not necessarily a multiple of 8 bits. In [14], each interface label has a fixed length and a hop counter is added to the header, allowing identification of the current path position. This counter is then decremented by each router before forwarding the packet involving data modifications. Similarly to [14], due to variable interface label sizes, a pointer is also needed in [15] to point to the current position in the encoded path. After reading the 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 packets on several interfaces. However, the header modifications performed by the router are only based on its local information; 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 [17], the source builds a Bloom filter based on the addresses of the nodes of the path, and stored in the packet header. At the reception of a packet, each router checks whether the addresses of its neighbors 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 positives while maintaining a reasonable size. Finally, for multicast transmissions, where 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) (to be read from the right to the left). A simple method to generate the encoded path is to concatenate the interface labels. More elaborated strategies presented in [18] and [16] encode the path into an integer number. The routers recover their information by computing the residue of this integer modulo a prime number.

3 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 the 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 ($011_b \oplus 111_b = 100_b$).

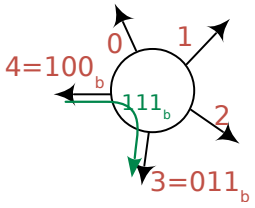

Fig. 1 Unicast interface label

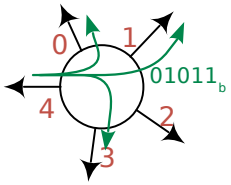

Fig. 2 Multicast interface label

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

The 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 obtain 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. In contrast to [15], the interface labels list does not need to be ordered in the path label, preventing the use of a pointer or vector. This filtering function is simply a few-dot products of short vectors that can be done on-the-fly, compliant with fast routing strategie such as, e.g., cut-through.

Briefly, 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.

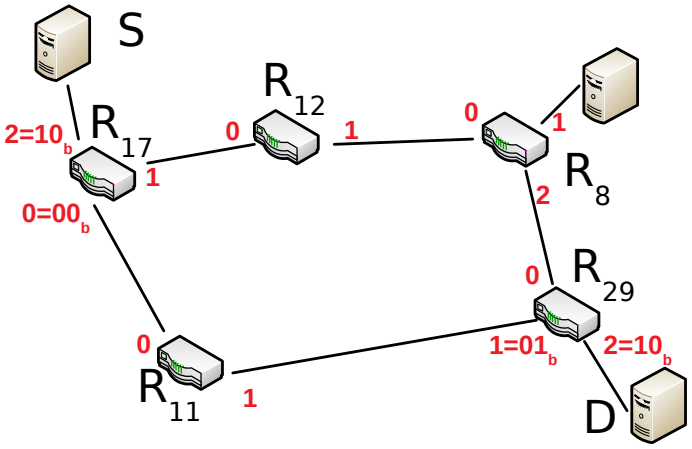

Fig. 3 Example network

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 the interface labels varies according to the routers or can be variable for the same router as in [15]. 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}:

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

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:

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

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

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

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:

$$P \cdot M = L \tag{6}$$

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

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

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

4 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.

4.1 Notations

The most important notations used in this paper are listed thereafter.

Table 1 Notations

R_i	defines a router with ID number i;
L_i	is an interface label for router R_i;
L	is a path label between a source S and a destination D. The concatenation of the interface labels correspond to a bit vector denoted $L = (L_{i_1},\ L_{i_2}, \ldots, L_{i_p})$ with a size $s_L = \sum_{k=1}^{p} s_{i_k}$;
P	is the length of the path label L;
$\overline{M_i}^{(e)}$	is the binary filtering stored matrix ID number e of router R_i defined by $\overline{s_P}$ rows and $\overline{s_i}$ columns;
$M_i^{(e)}$	is the binary filtering submatrix ID number e of router R_i defined by s_P rows and s_i columns;
$\overline{s_P}$	is the maximum length in bits of the size of P;
$\overline{s_i}$	is the maximum size of the interface labels of R_i;
M	is the concatenation of the filtering matrices used by the routers of the path.

4.2 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 in 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 $(R_{17}|R_{11}|R_{29})$ or $(R_{17}|R_{12}|R_8|R_{29})$. The connection can be unicast, multipath or multicast.

4.3 Router Forwarding

The main operation performed 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 Section 3 is to consider that each router stores 2^ϵ binary filtering matrices denoted $\overline{M}_i^{(e)}$, where $e = 0, \ldots, 2^\epsilon - 1$, as shown in Fig. 4. 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 (denoted step #1 in Fig. 4) and the value e set by the source stored on ϵ bits (denoted step #2). This value e is encoded on ϵ bits. In the following, the value of ϵ is set to 4. It means that 2^4 matrices will be stored numbered from 0 to 15. The number e, thus corresponds to the matrix that must be selected (step #3) to compute L_i (step #4). 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 that consist in multiplying the path label P by a filtering matrix M_i with s_i columns and s_P rows.

As a matter of fact, the size of ϵ impacts on the path label size and on the number of different filtering matrices stored by a router. We will discuss on the choice of this value in Section 5.

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

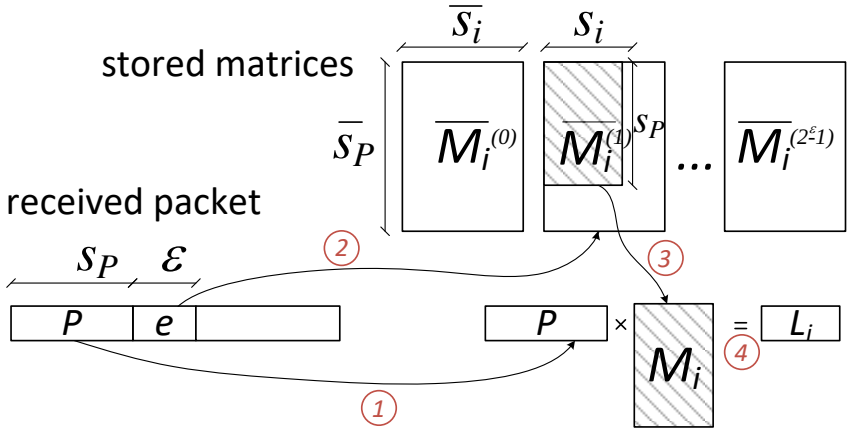

Fig. 4 Receiver operations. The numbers represent the different steps.

4.4 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 constraints. These constraints 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 demonstrate this theorem, it is sufficient to prove that if a router receives a packet with a path label P on an input interface numbered ID_i, forwarded via the output interface ID_o, then, the resulting feedback packet incoming on the same interface ID_o will be forwarded on the same interface ID_i using the same path label.

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 3, L_u is the XOR of the input interface ID_i and the output interface ID_o. Let's identify these interfaces as forward path interface as follows: ID_i^f and ID_o^f. Since ID_i^f is known, we can recover the ID_o^f by XORing L_u and ID_i^f as follows:

$$ID_i^f \oplus L_u = ID_i^f \oplus (ID_i^f \oplus ID_o^f) = ID_o^f$$

Following this operation, the packet is forwarded on this output interface ID_o.

On the reverse path, it receives a feedback packet on the return interface denoted ID_i^r with the same path label. By applying the filtering function M_u, it recovers L_u. Similarly, it computes $ID_i^r \oplus L_u$ to obtains ID_o^r as follows:

$$ID_i^r \oplus L_u = ID_o^r$$

as $ID_i^r = ID_o^f$, in other words, the forwarding output port is equal to the feedback input port, we then have:

$$ID_i^r \oplus L_u = ID_o^f \oplus L_u = ID_i^f$$

□

5 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 4.3. Note that we make no assumption about the filtering matrices and consider them as random binary matrices.

5.1 Probability of Construction of a Valid Path Label

According to 4.4, a valid path label can be built if the matrix M is invertible. [19] shows that the probability that a $s_L \times s_L$ binary random matrix is invertible is equal to

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

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

A valid path can be built 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 in 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).

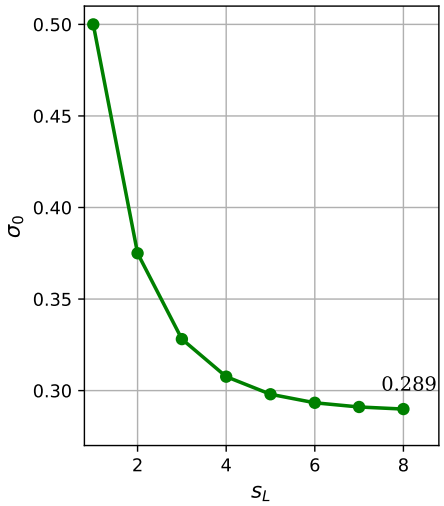

Fig. 5 proba. invertible matrix

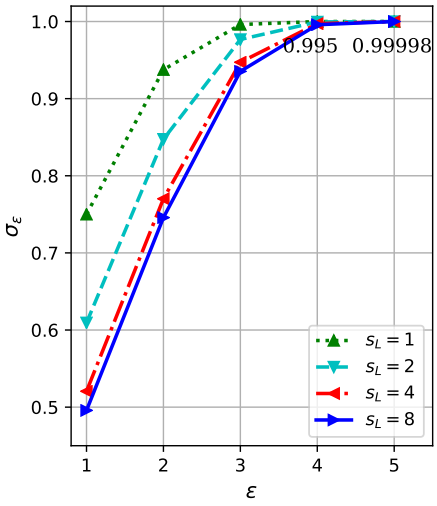

Fig. 6 proba. valid path

5.2 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 [20] 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 $k-1$ fails 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:

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

5.3 Number of Signalling Bits

Section 5.1 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 [15] which is 4 bits in most of the studied configurations.

5.4 Storage Amount in Routers

According to 4.3, each router stores 2^{ϵ} binary matrices of $\overline{s_P}$ rows and $\overline{s_i}$ columns. Therefore, 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 8.2). In the largest case studied, the path label has a size of approximately 200 bytes and the interface labels have a maximal size of approximately 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 to reduce the value of ϵ. Another possibility is to use filtering matrices that can be deduced from a short representation as in 3 where the columns of the filtering matrices are deduced from the first column by cyclic permutations.

6 Extensions of XSR

In this section, we propose two extensions of XSR. The first extends the principle to more general finite field. Since data packets usually carry bits, we only consider binary extension fields $\mathbb{F}_{2^w}$, *i.e.* extension fields of $\mathbb{F}_2$, but the concepts can be theoretically applied to any finite fields. Thus, considering the filtering function now defined as

follows:

$$F_i \begin{cases} \mathbb{F}_{2^w}^{s_P/w} & \longrightarrow \mathbb{F}_{2^w}^{s_i/w} \\ P & \longrightarrow P \cdot M_i \end{cases}$$

We need to update both theorems 1 and 2 proofs. The second extension is based on the observation that the construction of path is also possible from specific singular matrices.

Both extensions lead to a significant improvement of the probability of building valid paths.

6.1 Extension of XSR to finite fields $\mathbb{F}_{2^w}$

Theorem 1 gives a first solution to build a valid path label in the case where M is non-singular. In the following, we show that we can extend it to fields $\mathbb{F}_{2^w}$.

In practice, the device that wants to build the path (*i.e.,* the source or the network manager) first considers the matrix $M^{(1)}$ built from the first filtering matrices $M_{i_1}^{(1)}$, $M_{i_2}^{(1)}, \ldots, M_{i_p}^{(1)}$ stored by the routers of the path. If the matrix is not invertible, it considers $M^{(2)}$ built from the second matrices stored by the routers and so on until finding a matrix $M^{(j)}$ invertible. Then, it indicates in the packet field e (see previous paragraph) which matrix must be used to build/filter this path label. The number of stored matrices is chosen to ensure a path label construction with a targeted probability.

To evaluate the interest of using different finite fields, we first evaluate the probability of building a valid path with each finite field. To compare the different fields, we assume that the encoded vector L is a vector of $s_L = \sum_{k=1}^{p} s_{i_k}$ bits and that each s_{i_k} (and thus s_L) is divisible by w. For each possible w, we consider that a path vector of size s_L/w over $\mathbb{F}_{2^w}$ must be encoded from the $s_L/w \times s_L/w$ over $\mathbb{F}_{2^w}$.

According to Theorem 1, a valid path can be built if the encoding matrix M is invertible. Several studies have given the probability that a random square matrix of a given size defined over a given finite field is invertible.

Based on [21] (p. 338), reference [22] gives the probability $\mathcal{P}(n, q, r)$ that a random $n \times n$-matrix M defined over $\mathbb{F}_q$ has rank r :

$$\mathcal{P}(n, q, r) = \frac{1}{q^{n^2}} \begin{bmatrix} n \\ r \end{bmatrix}_q \sum_{k=0}^{r} (-1)^{r-k} \begin{bmatrix} r \\ k \end{bmatrix}_q q^{nk + \binom{r-k}{2}} \tag{12}$$

$$\text{where } \begin{bmatrix} n \\ r \end{bmatrix}_q = \prod_{l=0}^{r-1} \frac{q^{n-l} - 1}{q^{l+1} - 1}$$

According to Theorem 1, a valid path can be built with probability $\mathcal{P}_1(n, q) = \mathcal{P}(n, q, n)$. Fig. 7 is obtained with (12) by varying s_P as a function of a set of w values.

It is clear that the size of the finite field has a big impact of the probability to build a valid path. It follows that it has an impact on the number of matrices needed

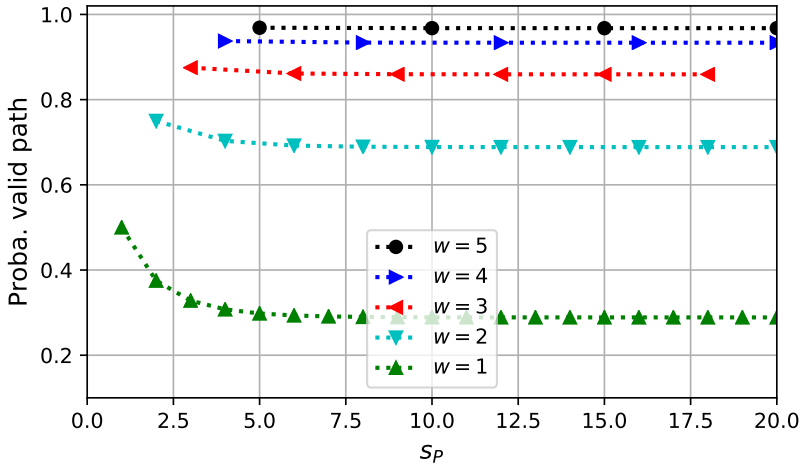

Fig. 7 Probability of building a valid path in $\mathbb{F}_{2^w}$ with Theorem 1

to be stored by each router to obtain a global targeted probability of building a path (discussed in Section 5 and on the number of signaling bits ϵ.

Theorem 2 is clearly valid for fields $\mathbb{F}_{2^w}$ because it depends solely on the representation of the interfaces and not on the types of operations performed to recover the output interface.

6.2 Building Path Labels With Singular Matrices

Even the use of binary extension fields increases the probability of building a valid path. It is sometimes difficult to use a large finite field because the size of the field is linked to the number of routers interfaces. In the following, we show that a valid path can sometimes be built even if the matrix is non-invertible.

6.2.1 Path Label Construction

Let us consider M a $s_L/w \times s_L/w$-singular matrix over $\mathbb{F}_{2^w}$ with rank $r < s_L/w$. Then $s_L/w - r$ columns can be expressed as linear combinations of r columns that are linearly independent. Without loss of generality, we can assume that the r first columns of M are linearly independent and that the $s_L/w - r$ other ones are linear combinations of the r first ones. Similarly, since the first r columns of M are linearly independent, they contain a $r \times r$-submatrix invertible. To simplify the notations in the following, we assume, without loss of generality, that its first r rows are linearly independent, and thus, the matrix $M_{|r}$ defined as the $r \times r$-submatrix of M built from its r first rows and columns is invertible.

Let us denote by C_j the j^{th} column of M. We then have:

$$C_u = \sum_{k=1}^{r} a_{u,k} C_k \tag{13}$$

for $u = r+1, \ldots, s_L/w$ and $a_{u,k} \in \mathbb{F}_{2^w}$. Let us now consider L as a vector of s_L/w elements of $\mathbb{F}_{2^w}$ $(l_1, \ldots, l_{s_L/w})$. Then,

Theorem 3. *There exists a valid path label if and only if:*

$$l_u = \sum_{k=1}^{r} a_{u,k} l_k \tag{14}$$

for $u = r+1, \ldots, s_L/w$.

Under this condition, a valid path label P *is :*

$$(p_1, \ldots, p_r, \overbrace{0, \ldots, 0}^{s_L - r}) \tag{15}$$

where $(p_1, \ldots, p_r) = (l_1, \ldots, l_r) \cdot M_{|r}^{-1}$.

Proof. Let us first assume that there exists a valid path label P such that $P \cdot M_i = L$. (13) states that $C_u = \sum_{k=1}^{r} a_{u,k} C_k$ for $u = r+1, \ldots, s_L/w$. Then,

$$\begin{aligned} P \cdot C_u &= P \cdot \sum_{k=1}^{r} a_{u,k} C_k \\ &= \sum_{k=1}^{r} a_{u,k} P \cdot C_k \end{aligned} \tag{16}$$

Since $P \cdot M_i = L$, we have $P \cdot C_j = l_j$ for any $j = 1, \ldots, s_L/w$. It follows from (16) that $l_u = \sum_{k=1}^{r} a_{u,k} l_k$ for $u = r+1, \ldots, s_L/w$.

Now, by assuming that (14) holds, we should prove that there exists a solution. To do this, we will prove that the vector P given in (15) is a valid path label. For that, we only should prove that $P \cdot M_i = L$. Let us first consider the product of P by the $s_L/w \times r$-submatrix of M_i built from its r first column. As the $s_L/w - r$ last values of P are equal to 0, this product is equal to $(p_1, \ldots, p_r) \cdot M_{|r}$. Since $(p_1, \ldots, p_r) = (l_1, \ldots, l_r) \cdot M_{|r}^{-1}$, this product is equal to $(p_1, \ldots, p_r) = (l_1, \ldots, l_r) \cdot M_{|r}^{-1} \cdot M_{|r} = (l_1, \ldots, l_r)$. Thus, we have proved that the path label is valid for the first r values of L. To complete the proof, let us consider the product $P \cdot C_u$, for $u = r+1, \ldots, s_L/w$. Then,

$$P \cdot C_u = \sum_{k=1}^{r} a_{u,k} P \cdot C_k \overset{(a)}{=} \sum_{k=1}^{r} a_{u,k} l_k \overset{(b)}{=} l_u \tag{17}$$

Equality (a) is obtained from the first step of this proof on the first r components of L and (b) is obtained with the hypothesis that (14) holds. □

To evaluate the interest of Theorem 3, compared to the path construction of Section 4.4, we need to evaluate the increase in the probability of building a valid path.

Recall that $\mathcal{P}(n, q, r)$ is the probability that a $n \times n$-binary matrix defined over $\mathbb{F}_q$ has rank r. The global probability of finding a valid path label, denoted by σ, is equal to:

$$\mathcal{P}_2(s_L/w, 2^w) = \sum_{r=0}^{s_L/w} \frac{\mathcal{P}(s_L/w, 2^w, r)}{2^{w.(s_l/w-r)}} \tag{18}$$

Indeed, according to Theorem 3, a valid path can be built if the rank of the matrix is r and if the last $s_l/w - r$ values of L verify (14).

Figure 8 shows the probability of building a valid path by using non-invertible matrices by varying w and s_P, length of the path in bits.

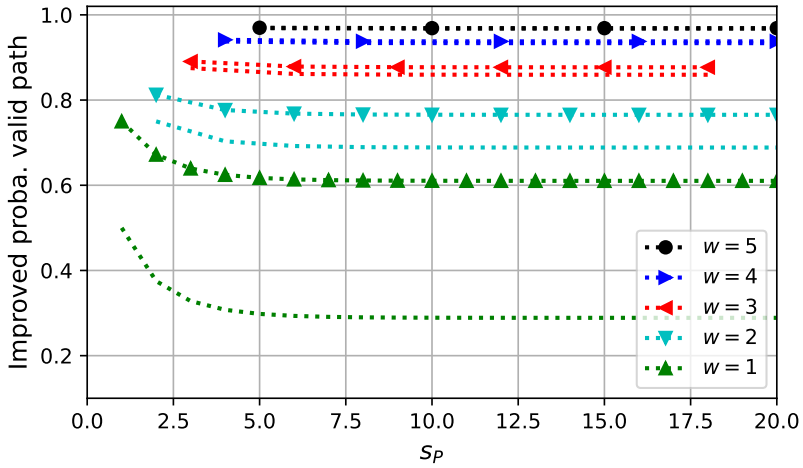

Fig. 8 Probability of building a valid path in $\mathbb{F}_{2^w}$ with Theorem 3

By comparing the results with the ones in Fig. 7 (recalled in dot lines), we observed significant gains for $w = 1$ and 2 and lower gains for higher values of w (which are already rather high).

7 Parameters Analysis over $\mathbb{F}_{2^w}$

We analyze in the following the router storage footprint of the filtering matrices, and then assess both the mathematical complexity of the filtering operation and the path construction.

7.1 Storage of the filtering matrices

From a system perspective, the filtering matrices must be known by both the filtering router and the entity that builds the path. Except for specific structured networks, these matrices are randomly chosen. A simple solution to store them is to generate them using a pseudo-random number generator (PRNG) from a seed shared by the routers. This solution can be used by the entity which generates the path because this operation is performed once at the beginning of the first connection between the sender and the destination. However, the duration of the execution of a PRNG would drastically slow down the forwarding process if computed at each packet arrival. Therefore, the filtering matrices must be generated once and then stored by the routers.

To evaluate this storage amount, we first need to evaluate the size of a matrix and then, the number of stored matrices.

Each filtering matrix stored by a router R_i has the size $\overline{s_i}/w \times \overline{s_P}/w$ elements of $\mathbb{F}_{2^w}$, so the number of bits stored is equal to

$$\frac{\overline{s_i}.\overline{s_P}}{w} \tag{19}$$

It can be observed that increasing the value of w reduces the amount of date stored for one matrix.

As explained in Section 4.4, the number n_M of matrices stored by the routers depends on the targeted probability $\mathcal{P}_3$ to build a path. We have :

$$\mathcal{P}_3 = 1 - (1 - \mathcal{P}_2(n, q))^{n_M} \tag{20}$$

If $\mathcal{P}_3$ is given, n_M can be deduced as follows :

$$n_M = \left\lceil \frac{\log(1 - \mathcal{P}_3)}{\log(1 - \mathcal{P}_2(n, q))} \right\rceil \tag{21}$$

It follows that the global amount of data (in bits) stored by a router is equal to :

$$\frac{n_M.\overline{s_i}.\overline{s_P}}{w} \tag{22}$$

For $\overline{s_i} = 4$ and $\mathcal{P}_3 = 1 - 10^{-6}$, Figure 9 evaluates this storage amount for a router for various values of $\overline{s_P}$ and w. This clearly demonstrates the interest of using finite fields larger than $\mathbb{F}_2$.

7.2 Complexity

Two different complexities must be considered. The most important one is the complexity of the filtering operation, which is critical to allow fast forwarding.

As explained in Section 4.3, this operation is a multiplication over $\mathbb{F}_{2^w}$ of a s_P/w-vector by a $s_P/w \times s_i/w$-matrix. So, the number of operations in the field is $s_P/w \times s_i/w$. Many efficient methods were proposed to optimize multiplications in finite fields

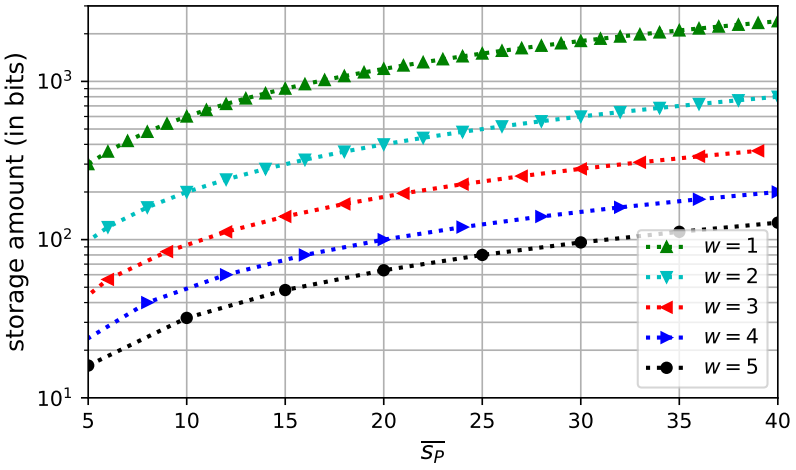

Fig. 9 Storage amount for $\overline{s_i} = 4$ and $\mathcal{P}_3 = 1 - 10^{-6}$

(see *e.g.*, [23],[24]), but it can generally be assumed that a multiplication in $\mathbb{F}_{2^w}$ can be done in $O(w^2)$ bit operations. It follows that the complexity of the filtering operation is in $O(s_P.s_i)$ number of bit operations and, thus, does not depend on w.

For the complexity of the path construction, the most costly operation is an inversion of a $s_P/w \times s_P/w$, which is in $O((s_P/w)^3)$ operations in $\mathbb{F}_{2^w}$, and thus, in $O(((s_P/w)^3.w^2) = O(s_P^3/w)$ bit operations. This complexity is thus reduced when w is increased. As explained in [25], the number of inversions is $1/\mathcal{P}_2(n, 2^w)$ which also decreases when w is increased. Therefore, using a large finite field also allows reducing the complexity.

7.3 Conclusion

To conclude the last two sections, the first important observation is that we obtain a better gain both in terms of complexity and storage within an extension of a finite field $\mathbb{F}_{2^w}$. However, the inherent simplicity of practically using $\mathbb{F}_{2^w}$ should lead to an implementation that is simpler to optimize considering the multiple bits operations available on CPUs today. As the gain depends on the motherboard specifications, the CPU, and the implementation itself, the choice of the best finite field should be done by commercial vendors following benchmarks on their own implementations. As a matter of fact, it would be inconsistent to propose any recommendation at this stage, without several vendors benchmarking over commercial hardware.

8 XSR Versus Existing Work and Application

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

8.1 Optimal Path Encoding for Unicast Transmissions

This first considered work, denoted OPE, is presented in [15]. The authors propose the use of prefix codes to represent the interface labels and optimize the choice of the labels 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 the router according to the length of its interface label. This scheme allows significantly reducing 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 number 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 of adding XSR on top of OPE is twofold:

1. the pointer used by OPE involves an 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 8.2). Encoding the path with XSR removes this notion of order and thus allows multicast transmissions;
2. using fast filtering router operations prevent any packet modification due to pointer update or possibly integrity checks.

8.2 Datacenter Networks

8.2.1 Recent Work in Source Routing for Datacenter Networks

The potential of source routing for datacenter networking was demonstrated in KeyFlow [26] and COXcast [18] 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 that encodes 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 [16]. 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 to reduce the size of the integer path. Unfortunately, the number 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 particularly in the context of multicast.

8.2.2 Path Length Comparison

The mechanism used in COXcast and RDNA is based on a concept similar to XSR. The main difference is that XSR is based on linear algebra, whereas both others are based on modular arithmetic. Linear algebra has 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 2 and 3;
- linear operations performed in routers are simple dot products and are less complex than modulo operations on integers;
- linear algebra provides 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 size. We consider the use cases studied in RDNA [16] and compute the corresponding header length for each solution.

The datacenter network analyzed in [16] is a 2-tier Clos network topology (shown Fig. 10) composed of two stages core switches (spine and leaf) and one stage of edge switches connected to hosts. Connections are defined between two hosts. The considered path is defined between the edge switches connected to the hosts source and destination. The longest path is from the edge switch connected to the source, to the edge switch connected to the destination through a first leaf, a spine and a second leaf.

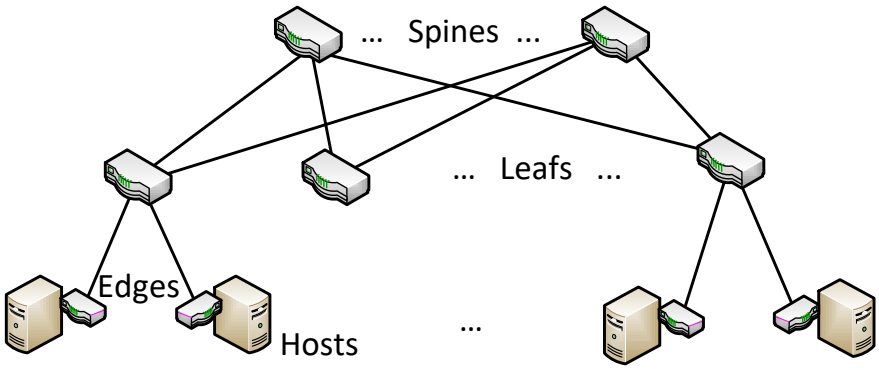

Fig. 10 Datacenter network

Table 2 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[18]	5	8	11	5	8	11	17	35	5	8	11	17	35	5	8	11	17	35
RDNA [16]	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

8.2.3 XSR - first method

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 4.4, 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:

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

The obtained values are compared to COXcast and RDNA in Table 2. 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 other leafs which forwards it to all their connected edge switches (see Fig. 4 of [16]).

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 4.4 leads to a encoded path of length $\texttt{ports}+\texttt{leafs}+(\texttt{leafs}-1).\texttt{ports}+\epsilon$ bits. By fixing the value of ϵ to 4 bits, we obtain the following number of bytes:

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

The values obtained are reported in Table 3 in the row "XSR #1". Except two cases, a small gain is observed in most configurations.

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

Table 3 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[18]	10	14	18	36	48	60	84	156	47	63	79	111	207	51	67	83	115	211
RDNA [16]	9	14	18	26	39	52	75	154	34	51	68	100	200	34	51	68	100	200
XSR #1	9	13	17	26	38	50	74	146	35	51	67	99	195	35	51	67	99	195
XSR #2	9	13	17	20	32	44	68	140	26	42	58	90	186	22	38	54	86	182

8.2.4 XSR alternative method - just another possible representation of the path label for multicast

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 [15] 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 other packets

This leads to an encoded path of length $2 + \texttt{ports} + \texttt{leafs} + (\texttt{leafs} - 1).(1 + \texttt{ports} - \texttt{spines}) + \epsilon$ bits. 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 perspective, 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 #2" of Table 3 show a significant gain in most use cases.

9 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 require 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.

References

[1] Internet Protocol. RFC Editor (1981). https://doi.org/10.17487/RFC0791 . https://rfc-editor.org/rfc/rfc791.txt

[2] Previdi, S., Filsfils, C., Decraene, B., Litkowski, S., Horneffer, M., Shakir, R.: Source Packet Routing in Networking (SPRING) Problem Statement and Requirements. RFC Editor (2016). https://doi.org/10.17487/RFC7855 . https://rfc-editor.org/rfc/rfc7855.txt

[3] Lee, T., Pappas, C., Basescu, C., Han, J., Hoefler, T., Perrig, A.: Source-based path selection: The data plane perspective. In: The 10th International Conference on Future Internet. CFI '15, pp. 41–45. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2775088.2775090 . https://doi.org/10.1145/2775088.2775090

[4] Abujoda, A., Kouchaksaraei, H.R., Papadimitriou, P.: Sdn-based source routing for scalable service chaining in datacenters. In: Mamatas, L., Matta, I., Papadimitriou, P., Koucheryavy, Y. (eds.) Wired/Wireless Internet Communications, pp. 66–77. Springer, Cham (2016)

[5] Parkes, S.M., et al.: SpaceWire: Links, Nodes, Routers and Networks. European Cooperation for Space Standardization, Standard No. ECSS- E50-12A (2003)

[6] Hu, Y.-C., Maltz, D.A., Johnson, D.B.: The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC Editor (2007). https://doi.org/10.17487/RFC4728 . https://rfc-editor.org/rfc/rfc4728.txt

[7] Li, S., *et al.*: Improving SDN scalability with protocol-oblivious source routing: A system-level study. IEEE Transactions on Network and Service Management **15**(1), 275–288 (2018) https://doi.org/10.1109/TNSM.2017.2766159

[8] Ventre, P.L., Salsano, S., Polverini, M., Cianfrani, A., Abdelsalam, A., Filsfils, C., Camarillo, P., Clad, F.: Segment routing: A comprehensive survey of research activities, standardization efforts, and implementation results. IEEE Communications Surveys & Tutorials **23**(1), 182–221 (2021) https://doi.org/10.1109/COMST.2020.3036826

[9] Guedrez, R., Dugeon, O., Lahoud, S., Texier, G.: A new method for encoding mpls segment routing te paths. In: 2017 8th International Conference on the Network of the Future (NOF), pp. 58–65 (2017). https://doi.org/10.1109/NOF.2017.8251221

[10] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., Shakir, R.: Segment Routing Architecture. RFC Editor (2018). https://doi.org/10.17487/RFC8402 . https://rfc-editor.org/rfc/rfc8402.txt

[11] Jadin, M., Aubry, F., Schaus, P., Bonaventure, O.: CG4SR: Near optimal traffic engineering for segment routing with column generation. In: IEEE INFOCOM, pp. 1333–1341 (2019)

[12] Giorgetti, A., Castoldi, P., Cugini, F., Nijhof, J., Lazzeri, F., Bruno, G.: Path encoding in segment routing. In: 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1–6 (2015). https://doi.org/10.1109/GLOCOM.2015.7417097

[13] Guedrez, R., Dugeon, O., Lahoud, S., Texier, G.: Label encoding algorithm for mpls segment routing. In: 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA), pp. 113–117 (2016). https://doi.org/10.1109/NCA.2016.7778603

[14] Soliman, M., Nandy, B., Lambadaris, I., Ashwood-Smith, P.: Source routed forwarding with software defined control, considerations and implications. In: Proceedings of the ACM Conference on CoNEXT Student Workshop, pp. 43–44. ACM, New York, NY, USA (2012). https://doi.org/10.1145/2413247.2413274

[15] Hari, A., Niesen, U., Wilfong, G.: On the problem of optimal path encoding for software-defined networks. IEEE/ACM Trans. Netw. **25**(1), 189–198 (2017) https://doi.org/10.1109/TNET.2016.2571300

[16] Liberato, A., *et al.*: RDNA: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters. IEEE Transactions on Network and Service Management **15**(4), 1473–1487 (2018) https://doi.org/10.1109/TNSM.2018.2876845

[17] Castelluccia, C., Mutaf, P.: Hash-based dynamic source routing. In: Networking 2004, pp. 1012–1023 (2004)

[18] Jia, W.: A scalable multicast source routing architecture for data center networks. IEEE Journal on Selected Areas in Communications **32**(1), 116–123 (2014) https://doi.org/10.1109/JSAC.2014.140111

[19] Berlekamp, E.R.: The technology of error-correcting codes. Proceedings of the IEEE **68**(5), 564–593 (1980) https://doi.org/10.1109/PROC.1980.11696

[20] Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**(4), 354–356

(1969) https://doi.org/10.1007/BF02165411

[21] Lint, J.H., Wilson, R.M.: A Course in Combinatorics, 2nd edn. Cambridge University Press, Cambridge (2001). https://doi.org/10.1017/CBO9780511987045

[22] Cooper, C.: On the distribution of rank of a random matrix over a finite field. Random Struct. Algorithms **17**(3-4), 197–212 (2000) https://doi.org/10.1002/1098-2418(200010/12)17:3/4⟨197::AID-RSA2⟩3.0.CO;2-K

[23] Blömer, J., Kalfane, M., Karp, R., Karpinski, M., Luby, M., Zuckerman, D.: An XOR-Based Erasure-Resilient Coding Scheme. Technical Report TR-95-048, International Computer Science Institute (1995)

[24] Detchart, J., Lacan, J.: Polynomial ring transforms for efficient xor-based erasure coding. In: 2017 IEEE International Symposium on Information Theory (ISIT), pp. 604–608 (2017). https://doi.org/10.1109/ISIT.2017.8006599

[25] Lacan, J., Lochin, E.: XOR-based source routing. In: 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR), pp. 1–7 (2020). https://doi.org/10.1109/HPSR48589.2020.9098991

[26] Martinello, M., Ribeiro, M.R.N., de Oliveira, R.E.Z., de Angelis Vitoi, R.: Keyflow: a prototype for evolving SDN toward core network fabrics. IEEE Network **28**(2), 12–19 (2014) https://doi.org/10.1109/MNET.2014.6786608