RDNA: Residue-Defined Networking Architecture Enabling Ultra-Reliable Low-Latency Datacenters

Alextian Liberato, Magnos Martinello, Roberta L. Gomes, Arash F. Beldachi, Emilio Salas, Rodolfo Villaca, Moisés R. N. Ribeiro, Koteswararao Kondepu, George Kanellos, Reza Nejabati, *Senior Member, IEEE*, Alexander Gorodnik, and Dimitra Simeonidou

***Abstract*—Datacenter (DC) design has been moved toward the edge computing paradigm motivated by the need of bringing cloud resources closer to end users. However, the software defined networking (SDN) architecture offers no clue to the design of micro DCs (MDCs) for meeting complex and stringent requirements from next generation 5G networks. This is because canonical SDN lacks a clear distinction between functional network parts, such as core and edge elements. Besides, there is no decoupling between the routing and the network policy. In this paper, we introduce residue defined networking architecture (RDNA) as a new approach for enabling key features like ultra-reliable and low-latency communication in MDC networks. RDNA explores the programmability of residues number system as a fundamental concept to define a minimalist forwarding model for core nodes. Instead of forwarding packets based on classical table lookup operations, core nodes are tableless switches that forward packets using merely remainder of the division (modulo) operations. By solving a residue congruence system representing a network topology, we found out the algorithms and their mathematical properties to design RDNA's routing system that: 1) supports unicast and multicast communication; 2) provides resilient routes with protection for the entire route; and 3) is scalable for 2-tier Clos topologies. Experimental implementations on Mininet and NetFPGA SUME show that RDNA achieves 600 ns switching latency per hop with virtually no jitter at core nodes and sub-millisecond failure recovery time.**

***Index Terms*—Network architecture, ultra-reliable, low-latency, datacenter.**

I. Introduction

CURRENT networks have evolved toward complex systems that are becoming too expensive, complicate to manage and highly susceptible to vendor lock-in. In this sense, the network research community has debated the "clean-slate" versus "evolutionary" approach designs for the Internet architecture [1]. Insights from both approaches can help guide the ongoing evolution of network architectures. Before adopting disruptive ideas, though, we need to provide incremental solutions to support specific forthcoming applications.

A recent research challenge is posed by the need of dealing with demands emerging from the next 5^{th} Generation (5G) mobile communications network. It is expected that networks evolve to support 5G requirements, having agility and dependability in provisioning connectivity across their processing elements, keeping in mind ultra-reliable and low-latency communications. Datacenters appear as the last element in the chain between end-users and services to provide end-to-end management, deployment capability for traffic engineering purposes, flexibility, and scalability.

For this new scenario, ITU has defined enhanced Mobile BroadBand (eMBB), Ultra-Reliable and Low-Latency Communication (URLLC), and massive Machine Type Communications (mMTC) [2]. Compared to the current network technology, 5G is expected to provide 100-fold increase in throughput (mainly for eMBB) and in the number of connected devices per km^2 (due to mMTC), whereas latency should be reduced 30-50 times, especially for URLLC clients.

To meet these new requirements, *Datacenter designs* have shifted to support edge computing (EC), emerging as a mean to bring cloud resources closer to end users [3]. Two basic goals motivate such architectural change: (i) offload the network core and the cloud from localized demands; and (ii) mitigate latency-related issues for real-time applications. In addition, edge computing modules are expected to provide Software as a Service (SaaS) to clients by having data processing in an always-on status and no need to wait for allocation of resources, software initiation, and specific configurations.

Caching and multicasting, data aggregation and analytics may also benefit from EC mediating traffic dynamics between air interfaces and cloud-based virtualized services. Beyond the expected human-oriented multicasting such as video applications, 5G will bring a discussion about ephemeral ad hoc multicasting from point-to-multipoint communication frequently present in mMTC [4].

Thus, the challenge to design a Datacenter network needs to consider at least two building blocks [5]: (i) the underlying

Manuscript received April 15, 2018; revised August 11, 2018 and October 10, 2018; accepted October 16, 2018. Date of publication October 18, 2018; date of current version December 10, 2018. This research has been supported by grants from CNPq, CAPES, FAPES, MCTI, CTIC, RNP, European Union's Horizon 2020 under grant agreement no. 688941 (FUTEBOL), and L020009 (TOUCAN). The associate editor coordinating the review of this paper and approving it for publication was G. Schembra. *(Corresponding author: Alextian Liberato.)*

A. Liberato, M. Martinello, R. L. Gomes, R. Villaca, and M. R. N. Ribeiro are with the Software-Defined Networks Laboratory, Federal University of Espírito Santo, Vitória 29075-910, Brazil (e-mail: alextian.liberato@ufes.br; magnos.martinello@ufes.br; roberta.gomes@ufes.br; rodolfo.villaca@ufes.br; moises.ribeiro@ufes.br).

A. F. Beldachi, E. Salas, K. Kondepu, G. Kanellos, R. Nejabati, A. Gorodnik, and D. Simeonidou are with the High Performance Networks Group, University of Bristol, Bristol BS8 1UB, U.K. (e-mail: arash.beldachi@bristol.ac.uk; e.huguessalas@bristol.ac.uk; g.kanellos@bristol.ac.uk; r.nejabati@bristol.ac.uk; a.gorodnick@bristol.ac.uk; dimitra.simeonidou@bristol.ac.uk).

Digital Object Identifier 10.1109/TNSM.2018.2876845

hardware representing the networking data plane and (ii) the software that controls the overall behavior of the network, representing the control plane. In this context, the Software Defined Networking (SDN) key concept is to decouple the networking data plane from the control plane: the latter holds the network intelligent decisions while the former merely hosts executive tasks based on tables to process incoming flows.

Unfortunately, in the current SDN architecture, there is no easy way to implement the functionalities to meet the requirements for 5G networks. We argue that this is so due to no clear distinction between two functional network parts: core and edge elements. There is no decoupling between the routing and the network policy, which means that when an SDN controller decides the actions to be applied to a flow, it also has to select a path for this flow, setting states on all the intermediate switches pro-actively or reactively (i.e., in a stateful approach). Moreover, an OpenFlow (the *de facto* standard for SDN) enabled switch is clearly far from a simple design, requiring to support lookups of hundreds of packets per second and complex actions that have to be specified by size limited multiple tables [6] hosted by expensive and power hungry Ternary Content Addressable Memories (TCAMs) [7].

In this paper, we propose RDNA (*Residue-Defined Networking Architecture*), an approach that explores the Residues Number System (RNS) [8] as a fundamental concept to facilitate and enhance network programmability. Differently from other works in [9]–[12], RDNA main contribution involves a *minimalist forwarding model* for the core nodes. Instead of forwarding packets based on traditional table lookup operations, the core nodes are tableless switches that forward packets using merely remainder of the division (modulo) operations. By solving residues congruence system from RNS, representing a network topology, we found out the algorithms and their mathematical properties to design RDNA. The key features of RDNA are described below:

- *Tableless Packet Forwarding:* every switch in the RDNA processes packets based on a simple and deterministic *modulo* operation, rather than looking up for an entry per potential destination. A route between a pair of hosts in RDNA is defined as the remainder of the division between a route-ID and a set of local switch-IDs.
- *Source Routing for Traffic Engineering:* Unlike the conventional hop-by-hop routing, based on universal or hierarchical target addresses, source routing uncouples the routing logic from data plane which simplifies forwarding elements offering *source route control* for traffic engineering.
- *Protection Mechanism by Programmable Residues:* Source routing takes long time to recover from failures [13], RDNA addresses this problem rerouting packets directly in the data plane as a faster alternative to controller-based route restoration. We demonstrate a protection mechanism along the entire route with an extremely fast failure reaction by using *programmable residues forwarding paths*. As soon as a core switch detects a link failure, it triggers an emergence precomputed route, which is already embedded into the packet header.

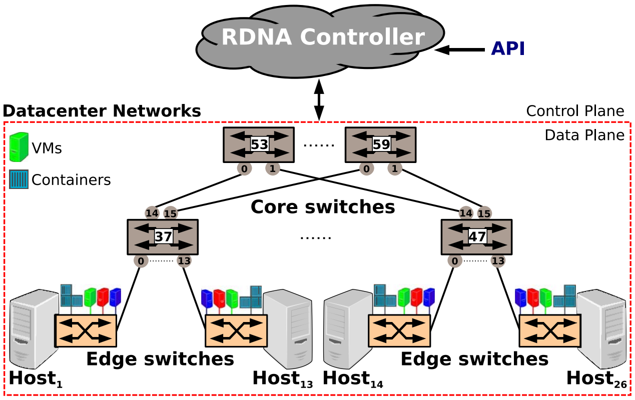


Fig. 1. Design of RDNA.

- *Multicast Communication:* This is the major extension added to our previous work [14], RDNA supports multicast communication by computing the residues relying on polynomial encoding, which is more scalable than classical congruence system used in other related works, such as [15] and [16].
- *Scalable for 2-Tier Clos Networks:* We carry out an RDNA scalability analysis considering *2-tier Clos network* topologies as a reference. This topology covers the majority of Enterprise Datacenter (EDC) or Micro Datacenter (MDC) deployments [3] to achieve efficient processing in EC applications [17].

The rest of the paper is organized as follows. In Section II we present the main concepts of RDNA and its design. In Section III we analyze the scalability of RDNA and benchmark it against other traditional approaches in literature. Section IV presents RDNA implementation in an emulated environment (Mininet) and in an experimental testbed using NetFPGA SUME. Section V discusses related work and Section VI summarizes this paper with our conclusions, final remarks and future research directions.

II. RDNA: Residue-Defined Networking Architecture

Built upon the principles of SDN, RDNA is composed by the following elements: i) the RDNA Controller, a *logically-centralized controller* which defines the programmable network configuration and its policies; ii) the *Edge switches*, that embed a compact encoding route-ID into packets; and iii) the *Core switches*, which are tableless and forward packets by computing a modulo operation from a route-ID information embedded into each packet. Thus, our architecture makes a clear separation between the edge and core switches to form two foundational blocks towards a pragmatic SDN design pattern. This separation allows to push the complexity to the network edge while keeping the network core extremely simple. Figure 1 illustrates the main elements of RDNA.

RDNA Controller: It is a central entity that is aware of the complete network topology, which is discovered by using the Link Layer Discovery Protocol (LLDP). This component is responsible to: (i) define the policy, assigning functions (e.g., route protection) to specific flows; (ii) select network route for each flow; (iii) calculate the route-ID (R_i) between each pair of hosts (or even virtual machines) and; (iv) send these R_i information to the edge switches in order to install it in flow packets. Besides, it can provide an Application Programming Interface (API) to support communication with external requests.

Core Switches: The idea behind the core switches is to replace the traditional lookup table operation by a tableless forwarding mechanism. The core switches operate only using *residues operations*, i.e., remainder of the division, based on switch-IDs and on the route-ID information embedded into packets. The switch-IDs, denoted by (S_i), are received from the RDNA Controller during the bootstrap phase. The set of S_i are not arbitrary numbers. They need to be:

- Pairwise primes, that is, any pair of numbers that has no common positive divisors other than 1.
- Larger than the number of physical interfaces on the respective core switch.

These simple requirements allow us to provide a unique route-ID (R_i) per path across the RDNA-enabled fabric. R_i calculation exploits RNS properties and is detailed later in Section II-A.

Edge Switches: They are the elements in charge of storing flow states and mapping the R_i received from the RDNA Controller into flow packets. This mapping means that a R_i has to be embedded into flow packets by the edge element. For example, if edge switches are OpenFlow enabled, fine grained control entries (per-flow states) should be installed on it.

Either a reactive or a proactive approach can be used here. In the first case, rules are installed when the ingress edge switch forwards the first packet to the RDNA Controller via *packet-in* operations requesting the route mapping. In the second case, the RDNA Controller pre-computes the route and installs in advance the incoming per-flow state via *flow-mod* for a route between source and destination.

A. RDNA Routing System

We formally define the RDNA network domain as a set *S* of *n* switches in a desired path, so that $S = S_i | i = 1, 2, \ldots, n$. Let *P* be a set of outgoing ports $P = \{p_1, p_2, \ldots, p_k\}$, where p_i is an arbitrary integer representing a physical port number (i.e., the outgoing port for the flow packets) on a switch S_i.

In order to make the congruence system solvable, *n* integers $S_1, S_2, \ldots, S_n$ need to be pairwise relatively primes. Then, there exists a unique integer *R* such that $0 \leq R < \prod_{i=1}^{n} S_i$ that solves the congruence system shown in Equation (1):

$$\begin{aligned} &< R >_{S_1} \equiv p_1 \\ &< R >_{S_2} \equiv p_2 \\ &\qquad \vdots \\ &< R >_{S_n} \equiv p_k \end{aligned} \tag{1}$$

We can rewrite Eq. (1) in a simple form as $< a >_b \triangleq$ *a modulo b*. For example, for $a = 12$ and $b = 5$, we have $< 12 >_5 = 2$, hence, 2 is the remainder of the division between 12 and 5.

Let *M* be

$$M = \prod_{i=1}^{n} S_i \tag{2}$$

The *Chinese Remainder Theorem (CRT)* [18], which is the basis of RDNA routing system, states that it is possible to reconstruct *R*, calculated through its residues in a RNS [8], as:

$$R = < \sum_{i \in S} p_i \cdot M_i \cdot L_i >_M \tag{3}$$

where,

$$M_i = \frac{M}{S_i} \tag{4}$$

$$L_i = < M_i^{-1} >_{S_i} \tag{5}$$

Eq. (5) means that L_i is the modular multiplicative inverse of M_i. In other words, L_i is an integer number such that:

$$< L_i \cdot M_i >_{S_i} = 1 \tag{6}$$

In terms of computation complexity, the CRT algorithm is $\mathcal{O}(len(M)^2)$, as demonstrated in [19], where *M* is calculated by Equation (2).

B. RDNA Encoding for Unicast Communication

To illustrate this concept, consider the scenario shown in Figure 2, where $host_1$ wishes to communicate with $host_{56}$. By using any routing algorithm (not shown here for the sake of clarity), the RDNA Controller selects an end-to-end path across the network according to the EDC policies, as presented in Figure 2 (**Step I**). For instance, it chooses the route to be set through the switches S = {37, 53, 47} composing what we call the primary route. In this case, the switches' output ports are P = {14, 3, 13}. Then, it computes a Primary Route Identification (PRI), e.g., *PRI* = 86446. The RDNA Controller sends the route-ID to the edge switches to install it in their respective flow-tables. Subsequently, the ingress edge switch is responsible for embedding PRI into each packet (e.g., into one of its header fields) coming from *src* host to *dst* host (**Step II**).

Once the packet has entered into the core, at every switch the packet arrives to, the remainder of the division between the packets' PRI (R = 86446) and the respective *switch-ID* is computed in order to define the appropriate output port to forward it. Thus, as shown in Figure 2, when S_{37} receives a packet with route-ID (R = 86446), it forwards the packet to port $< 86446 >_{37} = 14$ (**Step III**); then, S_{53} forwards it to port $< 86446 >_{53} = 3$ (**Step IV**); after, S_{47} forwards it to port $< 86446 >_{47} = 13$ (**Step V**), reaching the egress edge switch that removes the route-ID from the packet (**Step VI**) and delivers it to *dst* host (as can be seen in Figure 2).

C. RDNA Encoding for Resilient Routing

In the case of link failure, one traditional approach is **route restoration**. It consists on notifying the controller to recalculate the route excluding the faulty link from the available paths. The problem is how to react quickly after a failure detection, avoiding the intrinsic latency to communicate with

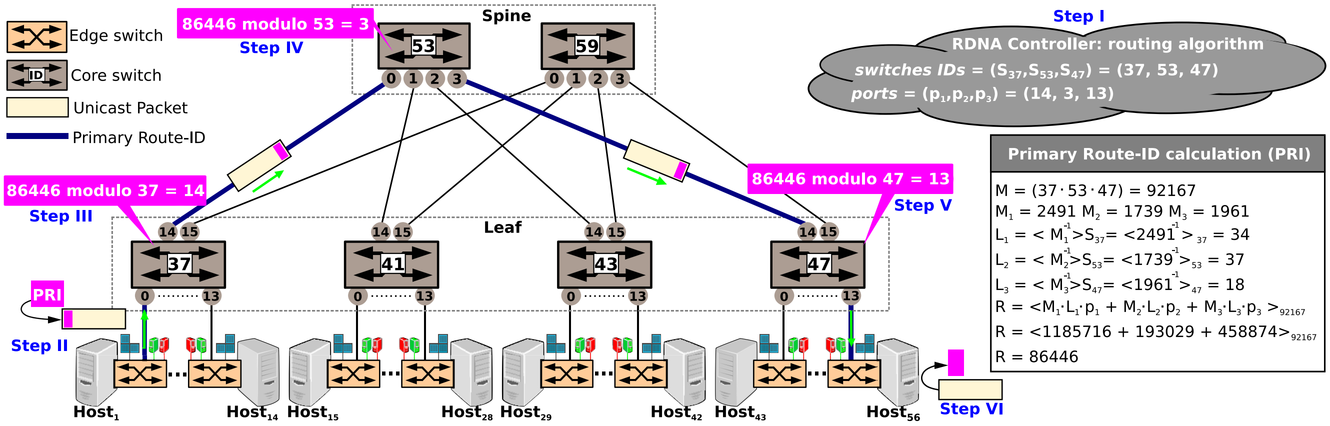

Edge switch
Core switch
Unicast Packet
Primary Route-ID
86446 modulo 53 = 3
Step IV
Spine
53
59
0 1 2 3
Step I
RDNA Controller: routing algorithm
switches IDs = (S37,S53,S47) = (37, 53, 47)
ports = (p1,p2,p3) = (14, 3, 13)
Primary Route-ID calculation (PRI)
M = (37·53·47) = 92167
M1 = 2491 M2 = 1739 M3 = 1961
L1 = < M1-1>S37= <2491-1> 37 = 34
L2 = < M2-1>S53= <1739-1>53 = 37
L3 = < M3-1>S47= <1961-1>47 = 18
R = <M1·L1·p1 + M2·L2·p2 + M3·L3·p3 >92167
R = <1185716 + 193029 + 458874>92167
R = 86446
86446 modulo 37 = 14
Step III
86446 modulo 47 = 13
Step V
Leaf
14 15
37
41
43
47
0 13
PRI
Step II
Host1
Host14
Host15
Host28
Host29
Host42
Host43
Host56
Step VI

Fig. 2. RDNA Routing System.

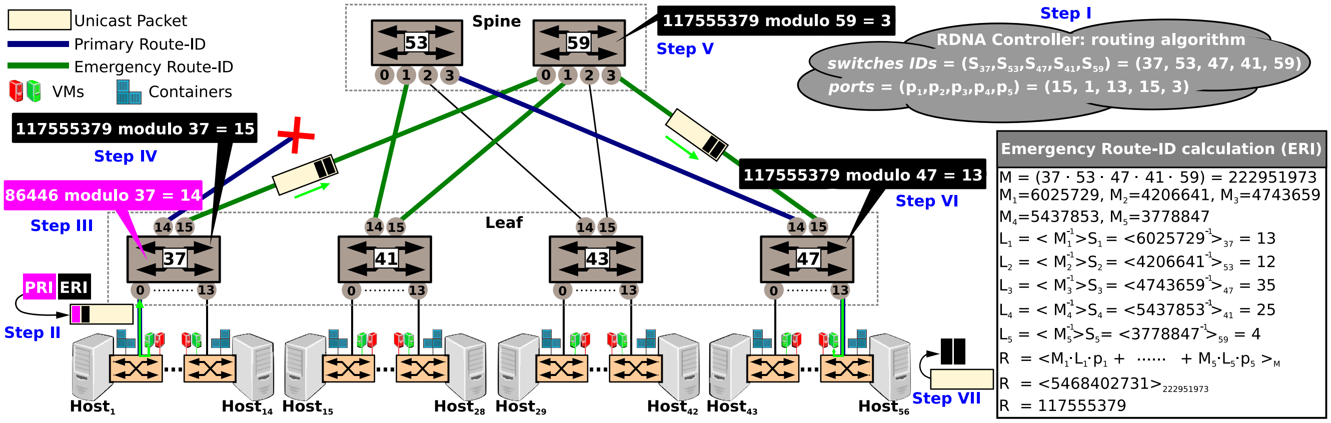

Unicast Packet
Primary Route-ID
Emergency Route-ID
VMs
Containers
Spine
53
59
0 1 2 3
117555379 modulo 59 = 3
Step V
Step I
RDNA Controller: routing algorithm
switches IDs = (S37,S53,S47,S41,S59) = (37, 53, 47, 41, 59)
ports = (p1,p2,p3,p4,p5) = (15, 1, 13, 15, 3)
Emergency Route-ID calculation (ERI)
M = (37 · 53 · 47 · 41 · 59) = 222951973
M1=6025729, M2=4206641, M3=4743659
M4=5437853, M5=3778847
L1 = < M1-1>S1 = <6025729-1>37 = 13
L2 = < M2-1>S2 = <4206641-1>53 = 12
L3 = < M3-1>S3 = <4743659-1>47 = 35
L4 = < M4-1>S4 = <5437853-1>41 = 25
L5 = < M5-1>S5= <3778847-1>59 = 4
R = <M1·L1·p1 + ······ + M5·L5·p5 >M
R = <5468402731>222951973
R = 117555379
117555379 modulo 37 = 15
Step IV
86446 modulo 37 = 14
Step III
117555379 modulo 47 = 13
Step VI
Leaf
14 15
37
41
43
47
0 13
PRI ERI
Step II
Host1
Host14
Host15
Host28
Host29
Host42
Host43
Host56
Step VII

Fig. 3. RDNA architecture design: fast failure reaction.

the controller. A typical mechanism for fast failure reaction is **route protection** through deflection. Deflection routing techniques are conceptually simple and allow every switch to independently decide which packets to forward to any available link [20].

Deflection routing is a probabilistic routing technique that may form transient loops [20]. To overcome this drawback, RDNA offers deterministic resilient routing based on a network protection mechanism along the whole route. Our fast failure reaction mechanism uses an Emergency Route Identification (ERI). ERI is computed to represent a set of switches necessary to bypass a failure in the primary route.

To illustrate this concept, consider the scenario shown in Figure 3. As in the previous scenario, $host_1$ wishes to communicate with $host_{56}$. So, Steps I, II and III are repeated, but, as part of the **Step I**, the RDNA Controller has also calculated the ERI as a unique value that composes a protection route for the whole primary route. In this example, we have $ERI = 117555379$, which sets the switches $S = \{37, 53, 47, 41, 59\}$ (with their output ports $P = \{15, 1, 13, 15, 3\}$) as the protection route. Thus, both PRI and ERI needs to be embedded into incoming packets at the edge switch.

Now, in this scenario, there is a failure link between S_{37} and S_{53}. Before packet forwarding, switch S_{37} checks the connection link to the next hop. As it is not available, it must overwrite the current PRI with the ERI. Then, using the overwritten PRI, S_{37} recalculates the remainder of the division in order to properly forward the packet. In this example, using route-ID ($R = 117555379$), S_{37} forwards packet to port $< 117555379 >_{37}= 15$ (**Step IV**). When S_{59} receives a packet with route-ID ($R = 117555379$), it forwards the packet to port $< 117555379 >_{59}= 3$ (**Step V**). Then, when S_{47} receives a packet with route-ID ($R = 117555379$), it forwards the packet to port $< 117555379 >_{47}= 13$ (**Step VI**). Note that S_{59} and S_{47} are unaware of the re-routing. It works just like an ordinary packet forwarding. Finally, the packet reaches the egress edge switch, which removes the route-IDs from the packet (**Step VII**) and delivers it to the destination host.

As previously mentioned, this approach allows fast recovery for the whole primary route. For example, if a failure occurs in the link between S_{53} and S_{47} instead, S_{53} executes **Step IV**, thus, overwriting PRI and forwarding the packet to port $< 117555379 >_{53}= 1$. When S_{41} receives a packet with route-ID ($R = 117555379$), it forwards the packet to port $< 117555379 >_{41}= 15$. Then, when S_{59} receives a packet with route-ID ($R = 117555379$), it forwards the packet to port $< 117555379 >_{59}= 3$ (**Step V**). Finally, when S_{47} receives a packet with route-ID ($R = 117555379$), it forwards the packet to port $< 117555379 >_{47}= 13$ (**Step VII**), allowing the packet

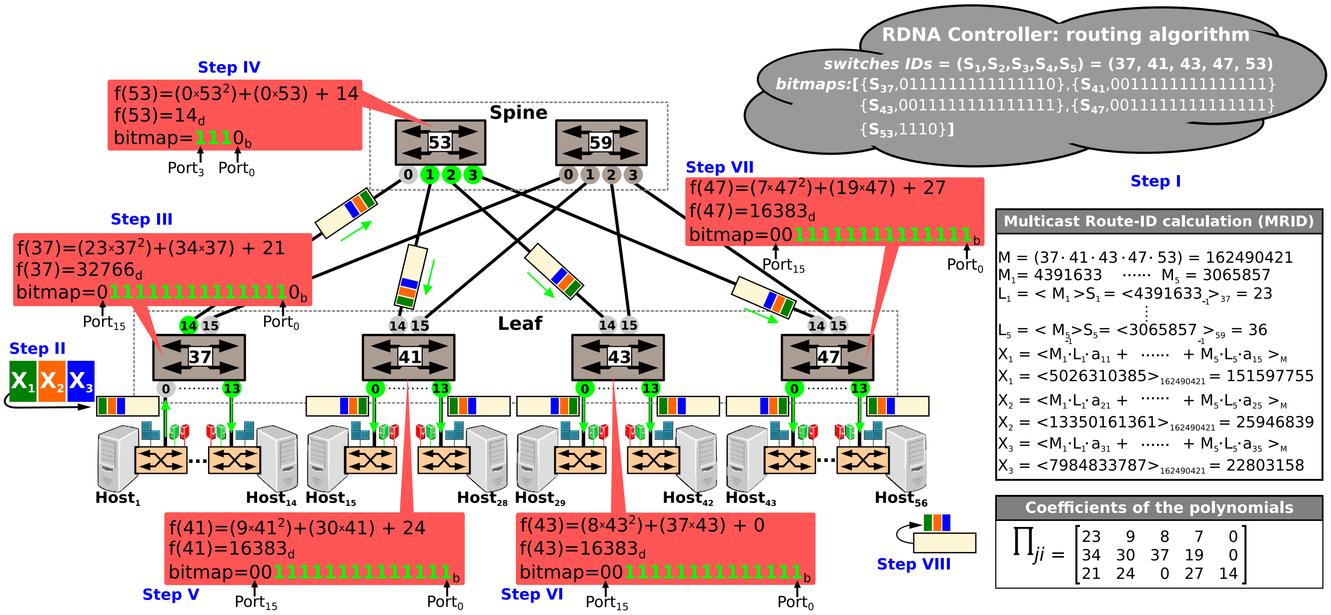

RDNA Controller: routing algorithm
switches IDs = (S1,S2,S3,S4,S5) = (37, 41, 43, 47, 53)
bitmaps:[{S37,0111111111111110},{S41,0011111111111111}
{S43,0011111111111111},{S47,0011111111111111}
{S53,1110}]
Step I
Multicast Route-ID calculation (MRID)
M = (37·41·43·47·53) = 162490421
M1 = 4391633 ...... M5 = 3065857
L1 = < M1 >S1 = <4391633 -1>37 = 23
L5 = < M5 -1>S5 = <3065857 -1>59 = 36
X1 = <M1·L1·a11 + ... + M5·L5·a15 >M
X1 = <5026310385>162490421 = 151597755
X2 = <M1·L1·a21 + ... + M5·L5·a25 >M
X2 = <13350161361>162490421 = 25946839
X3 = <M1·L1·a31 + ... + M5·L5·a35 >M
X3 = <7984833787>162490421 = 22803158
Coefficients of the polynomials
Πji = [23 9 8 7 0; 34 30 37 19 0; 21 24 0 27 14]
Step IV
f(53)=(0×53²)+(0×53) + 14
f(53)=14d
bitmap=1110b
Port3 Port0
Spine
53
59
0 1 2 3
0 1 2 3
Step III
f(37)=(23×37²)+(34×37) + 21
f(37)=32766d
bitmap=0111111111111110b
Port15 Port0
Step VII
f(47)=(7×47²)+(19×47) + 27
f(47)=16383d
bitmap=0011111111111111b
Port15 Port0
Step II
X1 X2 X3
Leaf
14 15
37
41
43
47
0 13
Host1 Host14 Host15 Host28 Host29 Host42 Host43 Host56
Step V
f(41)=(9×41²)+(30×41) + 24
f(41)=16383d
bitmap=0011111111111111b
Port15 Port0
Step VI
f(43)=(8×43²)+(37×43) + 0
f(43)=16383d
bitmap=0011111111111111b
Port15 Port0
Step VIII

Fig. 4. RDNA source routing multicast using RNS and polynomial expression over the 2-tier Clos Network topology.

to reach the edge switch, which removes the route-ID from the packet (**Step VII**) and delivers it to the destination host.

It is worth mentioning that the RDNA proposal does not really require core switches to be SDN enabled. The modulo operation and replacement of PRI by ERI are the only key functions that must be supported. Nevertheless, SDN would enable core switches to notify failures to controllers or to support dynamic switch-IDs registration and reconfiguration.

D. RDNA Encoding for Multicast Communication

In order to implement multicast routing, supporting one-to-many or many-to-many communication patterns, EDCs may use native IP multicast [21], [22]. However, this solution is not effective because switches might not reconfigure IP multicast groups at the required rates [23].

RDNA approach for multicast communication constructs the multicast tree based on the encoding of the bitmaps that represent the set of ports to which the packets must be forwarded to (on each switch). However, the fundamental problem in this case is how to select pairwise primes S_i such that each S_i must be larger than the number of possible combinations of output ports to construct the multicast tree. For instance, consider an EDC 2-tier Clos Network topology where $Spine = 2$ and $Leaf = 4$, with 16 ports per switch. In this case, we should find 6 primes larger than 2^{15} (the incoming port is not included).

In order to address this scalability problem, instead of using the modulo operation to directly compute the bitmaps for packet forwarding [15], [16], the modulo is used to get the coefficients of a specific polynomial function. The calculation of the polynomial function (to be made at **Step I**) and its coefficients are further detailed.

To explain how the RDNA multicast works, Figure 4 shows a green virtual machine (VM) instantiated on $host_1$. In this example, this VM wants to establish a multicast group allowing it to send data to all other green VMs (instantiated on the other physical hosts).

For the EDC of Figure 4, consider that three multicast identifiers, denoted by $X_1 = 151597755$, $X_2 = 25946839$, and $X_3 = 22803158$, were computed as part of **Step I**. On **Step II**, these identifiers are embedded into the packet, which is then forwarded to the first core switch S_{37}. As part of **Step III**, this switch calculates the modulo operation between X_i and S_{37}, obtaining $< 151597755 >_{37} = 23$, $< 25946839 >_{37} = 34$, and $< 22803158 >_{37} = 21$. Then, it calculates the polynomial function $f(37) = (23 \times 37^2) + (34 \times 37) + 21 = \mathbf{32766}$, which corresponds to the bitmap (0111111111111110_b), where the "1" bits specify the output ports for the packet to be forwarded.

When the packet arrives at S_{53} (Figure 4), the core switch reads its header and computes the module with the same X_1, X_2, X_3, obtaining $< 151597755 >_{53} = 0$, $< 25946839 >_{53} = 0$, and $< 22803158 >_{53} = 14$ (**Step IV**). Calculating the polynomial expression, the core switch S_{53} obtains $f(53) = (0 \times 53^2) + (0 \times 53) + 14 = \mathbf{14}$, which represents the bitmap (1110_b).

Still in Figure 4, when the packet arrives at S_{41}, S_{43} and S_{47}, the same process happens, but the bitmap must be (0011111111111111_b) to form the desired multicast tree. So, they calculate the module with the same X_1, X_2, X_3 obtaining their own coefficient values: for S_{41}, $< 151597755 >_{41} = 9$, $< 25946839 >_{41} = 30$, $< 22803158 >_{41} = 24$; for S_{43}, $< 151597755 >_{43} = 8$, $< 25946839 >_{43} = 37$, $< 22803158 >_{43} = 0$; and for S_{47}, $< 151597755 >_{47} = 7$, $< 25946839 >_{47} = 19$, $< 22803158 >_{47} = 27$ (**Step V, VI and VII**).

1) Computing the Polynomial Function and Its Coefficients: The first step to calculate the RDNA multicast encoding is to determine, at bootstrap phase, the polynomial degree and the S_i for every switch.

Pseudocode 1 Computing the Polynomial Degree

```
1: function DEGREE($S_i, p$)
2:     $fcoef \leftarrow S_i - 1$;
3:     $decimal \leftarrow 0$;
4:     $d \leftarrow 1$                                  ▷ Polynomial degree
5:     $maximum \leftarrow 2^{p-1}$
6:     $Seek \leftarrow False$
7:     while $Seek = False$ do
8:         $decimal \leftarrow decimal + (fcoef + (fcoef.(S_i^d)))$
9:         if $decimal \geq maximum$ then
10:            $Seek \leftarrow True$
11:        else
12:            $d \leftarrow d + 1$
13:    return $d$                            ▷ Polynomial degree found
14: end function
```

Pseudocode 2 Computing the Polynomial Coefficients

```
1: $coefvalues$ = Null;                      ▷ Coefficient list in $S_i$
2: function COEF($S_i, exponent, decimal$)
3:     $tmpvalue \leftarrow 0$;
4:     $limit \leftarrow (S_i - 1)$;               ▷ Limit for coefficient
5:     for $c \in range$(limit,-1, -1) do
6:         $tmp \leftarrow c \cdot (S_i^{exponent})$;
7:         if $decimal \geq tmp$ then
8:             $tmpvalue \leftarrow (decimal - tmp)$;
9:             Break;
10:    if $tmpvalue \geq 0$ then
11:        $coefvalues \leftarrow c$;
12:    else
13:        $coefvalues \leftarrow 0$;
14:    return $tmpvalue$
15: end function
16: function GET COEF($S_i, degree, bitmap$)
17:    $decimal \leftarrow int(bitmap, 2)$         ▷ Convert binary
18:    for $exp \in range$(degree,-1, -1) do
19:        if $(decimal < S_i)$ then
20:            if $(len(list) = degree)$ then
21:                $coefvalues \leftarrow decimal$;
22:                Break;
23:            else
24:                $coefvalue \leftarrow 0$;
25:        else
26:            $tempwish \leftarrow Coef(S_i, exp, decimal)$
27:            if $tempwish \geq 0$ then
28:                $decimal \leftarrow tempwish$
29:    return $coefvalues$                ▷ List coefficients
30: end function
```

The Pseudocode 1 presents a simple algorithm to get the polynomial degree *d*. This algorithm searches the minimum value of *d* (starting from 1) so that the polynomial result is high enough to represent the whole range of bitmaps at every core switch. So, for each switch, given its switch-ID (S_i) and the number of physical ports *p*, we compute the degree *d* using as coefficients the maximum possible value of the module for S_i (i.e., $S_i - 1$)[1] (**Line 2**). For instance, in the case of the core switch S_{37}, which has 16 ports, the degree 2 will satisfy this condition ($f(37) = (36 \times 37^2) + (36 \times 37) + 36 > 2^{15}$).

After getting the polynomial degree *d*, the RDNA controller also needs to compute the polynomial coefficients for each multicast tree to be configured on the topology. In the case of Figure 4, these coefficients are (a_3, a_2, a_1) from $f(i) = a_3 \times i^2 + a_2 \times i + a_1$. They will be used by every switch to calculate the resulting bitmap.

Pseudocode 2 shows our algorithm to figure out the coefficients for every S_i. In (**Line 16**), the RDNA Controller has S_i, the *degree* of the polynomial function and the required *bitmap* as its input. For instance, the algorithm receives as input 37, 2 and 0111111111111110_b. Then, starting from the largest degree (i^2), the algorithm searches for the maximum value of a_3, so that $(a_3 \times 37^2) \leq 32766 = 0111111111111110_b$. In this case, the value $23 \times 37^2 = 31487$ is less than 32766. Then 23 is added in the coefficient list *coefvalues*.

For the next term, (i^1), the same logic is applied, but using 1279 ($32766 - 31487$) as a limit. The algorithm searches for the maximum value of a_2, so that $(a_2 \times 37) \leq 1279$. In this case, the value 34, ($34 \times 37 = 1258$) is added to the coefficient list *coefvalues*, and the remainder 21 ($1279 - 1258$) is directly used as the last coefficient (i^0), ending the algorithm for the coefficients at S_{37}.

The same process occurs for S_{41}, S_{43}, S_{47} and S_{53}, and the resulting matrix is Π_{ji}, showed in Figure 4 (**Step I**). The columns represent the coefficients list for the set of core switches composing the multicast tree: S_{37}, S_{41}, S_{43}, S_{47}, and S_{53}, respectively.

The last step *is to compute* X_i, based on Chinese Remainder Theorem [18] (CRT). However, for the RDNA multicast communication, the output ports denoted by p_i are now replaced by the coefficients of the polynomial Π_{ji} that will produce the required bitmaps for S_i. Note that X_i represents the residue while L_i is the modular multiplicative inverse of M_i. This action concludes the RDNA encoding in **Step I** presented in Figure 4 as follows:

$$\begin{aligned}
M &= 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 = 162490421 \\
M_1 &= 4391633, M_2 = 3963181, M_3 = 3778847, \\
M_4 &= 3457243, M_5 = 3065857 \\
L_1 &= < 4391633^{-1} >_{37} = 23,\ L_2 = < 3963181^{-1} >_{41} = 20 \\
L_3 &= < 3778847^{-1} >_{43} = 37,\ L_4 = < 3457243^{-1} >_{47} = 36 \\
L_5 &= < 3065857^{-1} >_{53} = 14 \\
X_1 &= < L_1 \cdot M_1 \cdot \Pi_{11} + L_2 \cdot M_2 \cdot \Pi_{12} + L_3 \cdot M_3 \cdot \Pi_{13} \\
&\quad + L_4 \cdot M_4 \cdot \Pi_{14} >_M + L_5 \cdot M_5 \cdot \Pi_{15} >_M \\
X_1 &= < 4391633 \cdot 23 \cdot 23 + 3963181 \cdot 20 \cdot 9 \\
&\quad + 3778847 \cdot 37 \cdot 8 + 3457243 \cdot 36 \cdot 7 \\
&\quad + 3065857 \cdot 14 \cdot 0 >_M \\
X_1 &= < 2323173857 + 713372580 + 1118538712 \\
&\quad + 871225236 + 0 >_{162490421} = 151597755.
\end{aligned}$$

III. RDNA Scalability Analysis

In this section, we analyze the scalability of RDNA. Our basic assumption is that EDC network topology is based on 2-tier Clos networks, specially because the multi-stage Clos networks are topologies commonly found in micro Datacenters supporting tens of thousands of physical servers [17], [24], [25]. In a 2-tier Clos Spine-Leaf architecture, the number of uplinks from the leaf switches is equal

[1] The theorem of Euclidean division states that a remainder *r* of a division computation is an integer such that $0 \leq r < b$, where *b* is the divisor [13].

to the number of spine switches *v*. Thus, the total number of physical connections is the number of leaf switches *l* multiplied by the number of spine switches. Therefore, every lower-tier switch is connected to each top-tier switch in a full-mesh topology so that the RDNA controller may explore the *v* existing disjoint routes for load balancing and traffic engineering [26].

The scalability analysis is structured in three parts. The first part, described in Section III-A, consists of an analytical evaluation of RDNA scalability for EDC operating in unicast communications. The second (Section III-B) is devoted to evaluate the scalability of RDNA protection mechanism for failure recovery. In the third part, described in Section III-C, we tackle the RDNA scalability for multicast communications.

In order to benchmark RDNA to other existing schemes, COXcast approach [15] was selected because of its similarity to RDNA, relying on an equivalent source routing approach for Datacenter networks with tableless core switches. Basically, COXcast constructs the unicast path and multicast tree by encoding the corresponding output port bitmap of each intermediate node. The bitmap is obtained by the module operation using a common identifier and a node-specific key (i.e., prime number), so that the packets are routed to multiple receivers without requiring header modification. The difference is that RDNA uses the modulo operation to get the coefficients of a specific polynomial function, whereas COXcast uses the modulo operation directly to compute the bitmaps for packet forwarding.

A. Unicast Communication

For the sake of comparison, we assume the same set of topologies used in [15] which includes 23 different fan-outs of 2-tier Clos Networks. The length of a unicast path is computed between two hosts considering the shortest path ($n = 3$). The maximum number of Bytes required by RDNA encoding for unicast (R) can be computed as follows:

$$R = \frac{(\log_2(M = \prod_{i=1}^{n} S_i))}{8} \tag{7}$$

Equation (7) states that the higher the value of M, the larger the maximum required bit length. Recall that *PRI* is the Primary Route ID that varies as a function of the number of switches in the path n and switch-IDs (S_i).

As shown in Table I, RDNA is significantly more scalable than COXcast, reducing in 4.5 times the header size for unicast communication, on average. For some cases, this reduction is even higher (from 35 to 4 Bytes). The reason for this is that COXcast requires large pairwise prime numbers for the switch-ID, which increases the packet overhead, as can be seen in some topologies having switches with more than 96 ports.

B. Unicast Communication With Failure Recovery

Our analysis assumes an end-to-end protection of the entire route, thus, any failure occurred at any point along the primary route (PRI), will cause traffic to be moved to an emergency route (ERI) until the primary route be re-established. Taking again the Figure 3, consider the *src* as $host_1$ and the *dst* as $host_{56}$. PRI is set using the switches $S = \{37, 53, 47\}$ with output ports $P = \{14, 3, 13\}$ and ERI is set using the switches $S = \{37, 53, 47, 41, 59\}$ with output ports $P = \{15, 1, 13, 15, 3\}$.

TABLE I
HEADER SIZE (BYTES) FOR TYPICAL EDC CONFIGURATIONS USING 2-TIER CLOS NETWORKS

2-tier setting	Ports	Physical hosts	RDNA (PRI)	RDNA (ERI)	COXcast
Spine = 02 Leaf = 04	16	56	2	4	5
	24	88	4	7	8
	32	120	2	4	11
Spine = 04 Leaf = 08	16	96	2	4	5
	24	160	3	5	8
	32	224	3	4	11
	48	352	3	5	17
	96	736	3	5	35
Spine = 06 Leaf = 12	16	120	3	5	5
	24	216	3	5	8
	32	312	3	5	11
	48	360	4	6	17
	96	1080	4	6	35
Spine = 12 Leaf = 16	16	160	3	5	5
	24	288	3	5	8
	32	416	3	5	11
	48	672	4	6	17
	96	1440	4	6	35
Spine = 08 Leaf = 16	16	128	3	5	5
	24	256	3	5	8
	32	384	3	5	11
	48	640	4	6	17
	96	1408	4	6	35

As shown in Table I, RDNA is able to support failure recovery to protect the entire route with header size varying from 3 to 7 Bytes in all evaluated topologies. This full protection is a worst case scenario for **RDNA ERI** encoding, which corresponds to the maximum number of Bytes required by ERI (Equation (7)).

For full protection used by ERI, RDNA Controller needs to find the disjoint paths with the minimal hop-count. Thus, we have been inspired by an algorithm called Suurballe [27], whose computation time is twice greater than Dijkstra algorithm [28]. Suurballe algorithm allows the use of the same *Leaf* and *Spine* switches, as long as they have the same final host, as destination, through the union of common switches (S_i) into the emergency route.

Hence, if a failure happens in the link between S_{37} and S_{53}, S_{37} will use port 15 (ERI), then $S_{37} \rightarrow S_{59}$, $S_{59} \rightarrow S_{47}$ and $S_{47} \rightarrow dst$, which corresponds to one of the emergency routes encoded in ERI ($S = \{37, 59, 47\}$). If a failure occurs in the link between S_{53} and S_{47}, S_{53} will use port 1, then $S_{53} \rightarrow S_{41}$, $S_{41} \rightarrow S_{59}$, $S_{59} \rightarrow S_{47}$ and $S_{47} \rightarrow dst$, which corresponds to another emergency route encoded in ERI ($S = \{37, 53, 41, 59, 47\}$).

As can be seen in Table I, the RDNA encoding fits at the 12 Bytes of the Ethernet header. The PRI and ERI can be embedded in the packet (e.g., into the Ethernet header fields), thus, its bit length does not affect the total packet overhead. Assuming that Ethernet standard at the core is essentially for framing purposes, RDNA can use Ethernet header bits to encode PRI and ERI. As COXcast does not describe any failure recovery approach, it was not included in our analysis.

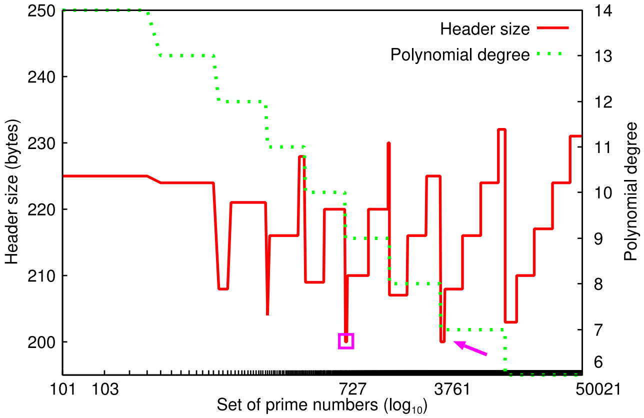


Fig. 5. Analysis of the set prime numbers for *Spine* = 08, *Leaf* = 16, *Port* = 96.

TABLE II
MAXIMUM OVERHEAD (IN BYTES) VERSUS NETWORK SIZE

2-tier setting	Ports	RDNA	COXcast	Xcast4	Xcast6
Spine = 02 Leaf = 04	16	9	10	12	1,008
	24	14	14	380	1,520
	32	18	18	508	2,032
Spine = 04 Leaf = 08	16	18	22	508	2,032
	24	27	30	764	3,056
	32	35	38	1,020	4,080
	48	54	54	1,532	6,128
	96	104	106	3,068	12,272
Spine = 06 Leaf = 12	16	26	36	764	3,056
	24	39	48	1,148	4,592
	32	52	60	1,532	6,128
	48	75	84	2,300	9,200
	96	154	156	4,604	18,416
Spine = 06 Leaf = 16	16	34	47	1,020	4,080
	24	51	63	1,532	6,128
	32	68	79	2,044	8,176
	48	100	111	3,068	12,272
	96	200	207	6,140	24,560
Spine = 08 Leaf = 16	16	34	51	1,020	4,080
	24	51	67	1,532	6,128
	32	68	83	2,044	8,176
	48	100	115	3,068	12,272
	96	200	211	6,140	24,560

C. RDNA Multicast Communication

For the multicast communication, the focus of our analysis is on understanding how RDNA multicast header is affected by the network size and the multicast group size. The scalability analysis for RDNA multicast covers 23 different configurations of 2-tier Clos network topology.

1) Impact of the Network Size: RDNA multicast relies on a polynomial function so that there is an optimal trade-off between the set of prime numbers S_i and the polynomial degree *d*. In order to find the optimal value, i.e., the smallest header size, we formulated the problem as a linear programming model, as follows:

$$\begin{aligned} \text{minimize} \quad & X_j^{d+1} = \frac{(\log_2(\prod_{i=1}^{n} S_i))}{8} \cdot (d+1) \\ \text{minimize} \quad & S_i \\ \text{subject to} \quad & \sum_{d=1}^{m} \left(S_i - 1 + \left(S_i - 1 \cdot \left(S_i^d \right) \right) \right) \geq 2^{p_i - 1} \\ & i, j, d \geq 1, \quad m \leq 14 \\ & S_i \in \mathbb{N} \quad | \quad i \quad is \quad prime > p_i, \end{aligned} \tag{8}$$

where the multi-objective function aims to minimize the header size X_j^{d+1} and S_i. The residues X_j^{d+1} are subject to $\sum_{d=1}^{m}$ that needs to be equal or greater than the maximum value for representing all the bitmaps $2^{p_i - 1}$, i.e., higher than the number of physical output ports p_i at switch S_i. Finally, we have limited the maximum polynomial degree m to 14 in order to limit the RDNA header size to the Ethernet header size.

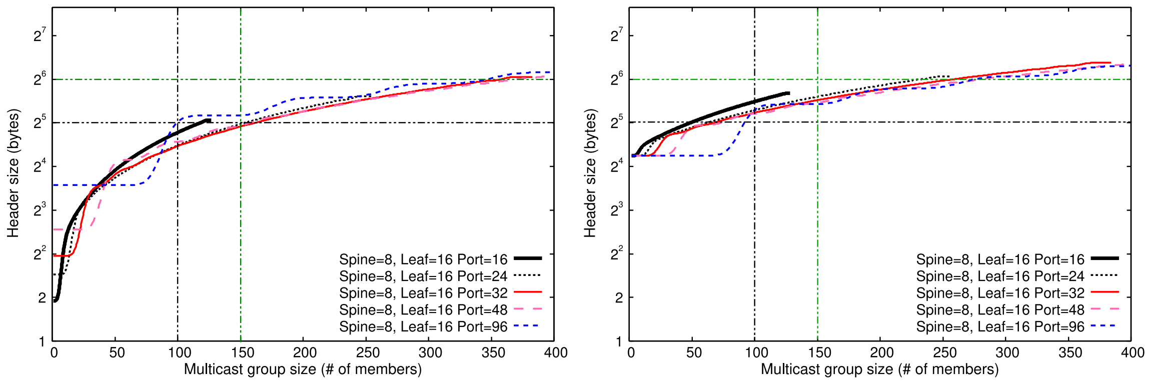
Figure 5 illustrates the optimization process, taking as example the largest network configuration, which is *Spine* = 08, *Leaf* = 16, *Port* = 96, limiting the *x*-axis to 50k (50021). For such network topology, this means that we have to select 24 S_i. However, the bigger is S_i (↑) the smaller is the polynomial degree *d* (↓). So if we go from the left to the right, the size of prime numbers increases as the polynomial degree decreases. Note that there are two minimal sets of 24 prime numbers that require 200 Bytes for the RDNA header size. The first choice starts at 727 [727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881] (pink square), and the second at 3761 [3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 3923, 3929, 3931] (pink arrow). Given that the S_i are lower in the first set, it is then chosen.

After the optimization process, we compare RDNA against the results presented in [15], that includes COXcast, Xcast4, and Xcast6 for various 2-tier Clos network sizes varying from *Spine* = 2, *Leaf* = 4 and *Port* = 16 until *Spine* = 8, *Leaf* = 16 and *Port* = 96.

As shown in Table II, as the network size grows, the multicast tree header scales well using occasionally 200 Bytes in the worst case, but it is mostly less than 100 Bytes. Moreover, RDNA multicast reduces the header size in all cases. On average, the RDNA reduces header sizes in 12% compared to COXcast, but it can reach up to 50% in some cases, e.g., *Spine* = 8, *Leaf* = 16 with *Port* = 16. This is due to the fact that COXcast needs to take huge prime numbers to represent the node-specific "key" in such way that the remainder of division gives the output ports as a bit array, not integers. The difference between COXcast and RDNA is significantly higher whenever the number of ports (node degree) is low.

2) Impact of the Multicast Group Size: We have assumed that the members of the multicast group are distributed to each container/VM placed in each physical host, given that an EDC has a non sparse distribution of the VMs (or containers) in their physical servers [29]. Hence, the multicast group size is incremented sequentially one-by-one until achieving all hosts in each network configuration.

For instance, in the RDNA multicast example, presented at Figure 4, $host_1$ is the source, then $host_2$ is added as group member, following by $host_3$, and so on and so forth until all the hosts have been included as group members. The results are depicted in Figures 6(a) and 6(b) for the largest network configuration.



(a)

(b)

Fig. 6. Required multicast header varying the number of ports to 2-tier Clos Network setting, where (a) RDNA multicast and (b) COXcast.

Figure 6(a) shows that RDNA header size is scalable as the multicast group increases for different number of physical ports. If the group size is set to 100 members, the header size keeps lower than 35 Bytes for all configurations. Increasing the group size to 150 members, only the configuration with 96 ports is larger than 32 Bytes (2^5). In other words, the increase of 50% in the multicast members leads to just a small increment in the header size. For instance, if we increase the group size to 350 members, the header size keeps below 64 Bytes but now with 3.5 times more members in the multicast group. Larger increments occur every time that a spine switch needs to be added to allow communication with a new leaf into the multicast group. Also, RDNA has substantially reduced the header facing up to COXcast, as presented in Figure 6(b). Comparing the size of groups lower than 100 members, RDNA header keeps smaller than 2^4 Bytes until 55 members, whereas COXcast needs 2^5 Bytes for all configurations. Larger increments in the header size occur for higher node degrees, e.g., a new leaf with 96 ports allows 88 members (96 − 8 leaf-spine connections) into the multicast group. [2]

To summarize, the results make evident that RDNA multicast is more scalable than the Xcast family so that RDNA has the potential to support the emerging demands for EDCs designs. In addition, we have seen that the RDNA multicast header is clearly more affected by the network size (specially when node degrees are high) than by the number of members in the multicast group, assuming that the multicast group distribution is not sparse.

IV. RDNA Implementation

For validation purposes, an RDNA prototype was implemented and evaluated. In this prototype, the RDNA Controller was developed as an application on top of Ryu [30]. Regarding the Edge switches, OpenFlow 1.3 enabled switches were used in the Mininet [31] emulation platform.

For the core switches, two prototypes of RDNA forwarding mechanism were implemented in two different platforms:

- Open vSwitch [32] (OvS), modifying its version 2.5.
- Field-Programmable Gate Array (FPGA) hardware using the NetFPGA SUME platform.

A. RDNA on Open vSwitch and Mininet Platform

When a packet arrives at the ingress edge switch, the packet is sent to the RDNA Controller which selects the primary and emergency routes among all pre-calculated paths between the source and destination. Then, the RDNA Controller installs OpenFlow rules at the ingress and egress switches.

As an example, Table III details the flow rules installed at the edges switches with the corresponding actions for setting a flow between $host_1$ and $host_{56}$, as previously illustrated in Figure 3. In this prototype, MAC address fields have been used to embed PRI and ERI into packets' header. So, in the ingress edge switch, the OpenFlow rule includes an action to add the route-IDs to packets' header, and an action to forward packets to the next hop in the core network. In the egress edge switch, the rule includes an action to rewrite the original MAC addresses, and an action to forward packets to the *dst* host. Flow entries at the edge switches are based on the destination IP address (plus optional VLAN or tenant identifiers).

The original Open vSwitch (OvS) implementation was modified to take advantage of the OpenFlow 1.3 Fast Failover [33] structure. Thus, when the switch receives a packet, it checks if the port is available, then PRI is used. However, if the switch port is not available, the core switch checks if PRI is different from ERI. If it is the case, it replaces PRI by ERI. From now on, the new route-ID is used to bypass the failed link. For the subsequent switches along the route, they just keep doing the modulo operation until the packets reach the edge. Only in the cases where another failure occur, packets will be forwarded to the RDNA Controller.

B. Impact of Fast Failure Reaction on TCP Throughput

In order to evaluate the efficiency of the recovery to link failures, we select different recovery techniques including reactive/proactive protection mechanisms. We aim to evaluate the granularity of failure recovery time and its impact on the TCP throughput. The experiments are carried out in Mininet where the network topology illustrated by Figure 3 was implemented. Each experiment was repeated 30 times and we plot

[2] $GroupSize = [(p - v) \times l] - 1$ where, p is the number of physical ports in each leaf switch, v is the number of spine switches, and l is the number of leaf switches, minus the source − 1.

TABLE III
FLOW TABLE ENTRIES IN THE EDGE SWITCHES

Flows Direction	Edge switches	Match	Action
Forward Flow	Ingress	MAC_DST: 00:00:00:00:00:56 and MAC_SRC: 00:00:00:00:00:01	SetField(eth_dst=00:00:00:01:51:AE, eth_src=00:00:07:01:C0:B3), output = 4
	Egress	MAC_SRC: 00:00:07:01:C0:B3 and IP_DST: 10.0.0.56	SetField(eth_dst=00:00:00:00:00:56, eth_src=00:00:00:00:00:01), output = 1
Backward Flow	Ingress	MAC_DST: 00:00:00:00:00:01 and MAC_SRC: 00:00:00:00:00:56	SetField(eth_dst=00:00:00:00:82:39, eth_src=00:00:07:37:2B:52), output = 4
	Egress	MAC_SRC: 00:00:07:37:2B:52 and IP_DST: 10.0.0.1	SetField(eth_dst= 00:00:00:00:00:01, eth_src= 00:00:00:00:00:56), output = 5

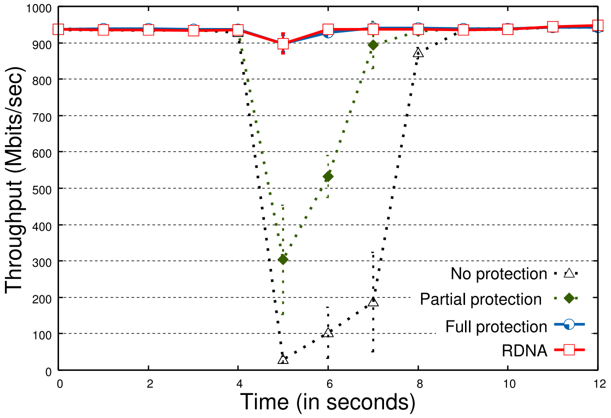


Fig. 7. Analysis of TCP throughput in Mbps for recovery time with different techniques and 95% confidence intervals.

the average results with a 95% confidence interval. During the experiments, links are disconnected using the Linux *ifdown* command to emulate link failures.

The failure recovery mechanisms considered for this analysis are the following:

- No protection: The controller waits for a switch link failure notification. After notification, it updates in a reactive mode the flow table rules for all the switches that belong to the selected path. This is our baseline.
- Partial protection: The controller installs proactively a set of protection entries only into the switches S_{37} and S_{53} using the OpenFlow Fast Failover (FF) resources [34]. In case of a failure on the primary route, a new entry should be added to switches S_{41} and S_{59} via controller.
- Full protection: OpenFlow FF rules are installed configuring all the switches to have an emergency route to protect the flow of a failure in any link of the entire route. This is the upper bound scenario on failure recovery time.
- RDNA: Flows are installed only at edge switches. Core switches react to a link failure just by replacing PRI by ERI rewriting the packets' header. Output ports are then computed according to the modulo operation of the ERI, steering the flow seamlessly through the emergency route.

Figure 7 presents the results (rate at 1 Gbps) for a failure in the $S_{37} \rightarrow S_{53}$ link (see Figure 3) comparing the different failure recovery mechanisms. Clearly, the worst case scenario is the reactive recovery as it depends on the communication with the control plane. Despite the reduction on failure recovery time by the Fast Failover, TCP throughput is drastically affected because the route had not been entirely protected. Moreover, although fast failover is an interesting local protection mechanism, there is additional burden to the administrators who have to add specific entries at every switch along the route, in both forward and backward paths.

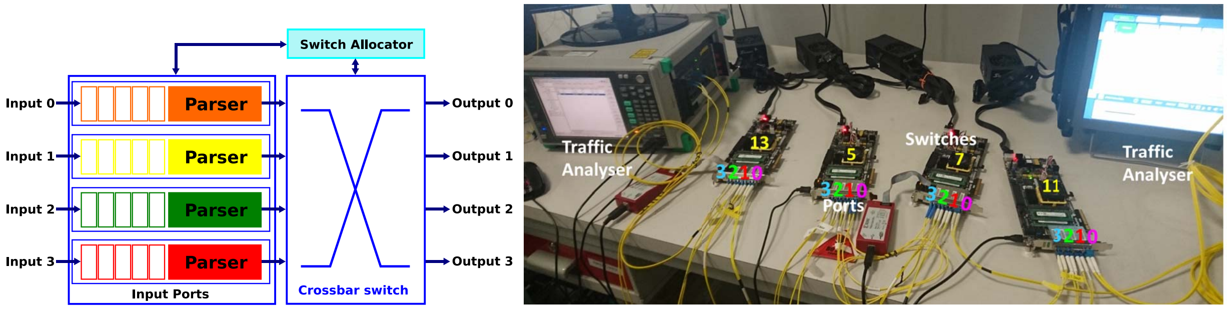
In contrast, the Full protection leads to a significant reduction on recovery time, not affecting the performance of TCP throughput. It is clear that RDNA has provided the same TCP performance as Full protection mechanism, but with much less complexity, since there is no need to add flow entries in every switch along the route.

Another important point is the limitation imposed by TCAM table size. Assuming an OpenFlow switch HP ProCurve J9451A [35], where flow tables of 1500 entries are available, for setting a protected flow is necessary 2 forward entries and 2 backward entries via OpenFlow group table resources FF for supporting the Full protection mechanism [34]. It means that only ($\frac{1500}{4} = 375$) simultaneous protected flows could be served. Therefore, considering the protection provided by OpenFlow FF features, only 375 VMs instanced on Datacenter take advantage of protection mechanisms. Though, RDNA core switch is tableless, and edge switches run as software, hence, they do not have such a strong constraint due to the flow-table sizes. Besides, this result has shown that (i) using packet rewrite actions at edge switches (insert and remove PRI and ERI) and (ii) triggering emergency routing only by the switch which has detected its local link failure; has enabled RDNA to offer equivalent carrier-grade failure recovery time.

C. RDNA Implementation on FPGA Platform

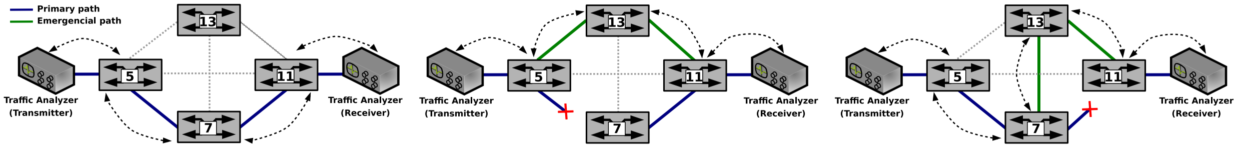
A previous NetFPGA implementation, but aimed at power consumption analysis at 1Gbps, for our tableless forwarding scheme can be found in [7]. Herein, Figure 8(a) shows the RDNA router architecture with four input/output ports implemented on a NetFPGA SUME. This implementation supports wormhole (WH) routing [36] which is a packet switching technique to reduce buffer space and latency. In WH switching, packets arriving at the input port are routed immediately to the output port as soon as the port is free. In this scenario, switch allocator is used to set the output ports based on route-ID (i.e., PRI or ERI) extracted from the packets by input ports parsers. Note that 10Gbps Ethernet MAC and PCS/PMA modules (Xilinx IP cores) are not represented in Figure 8(a).

The experimental setup is shown in Figure 8(b). It comprises 4 NetFPGA SUME boards to evaluate the core network in the proposed RDNA architecture, as previously illustrated in Figure 9. Each SUME FPGA board supports 4 optical



(a) (b)

Fig. 8. (a) Router architecture, (b) Experimental testbed setup 10Gbps.



(a) (b) (c)

Fig. 9. Topology full mesh with 4 core switches: (a) without failure (3 hops); (b) link failure S_5 to S_7 (3 hops) and (c) link failure S_7 to S_{11} (4 hops).

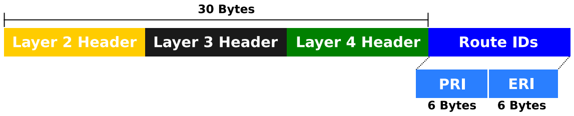
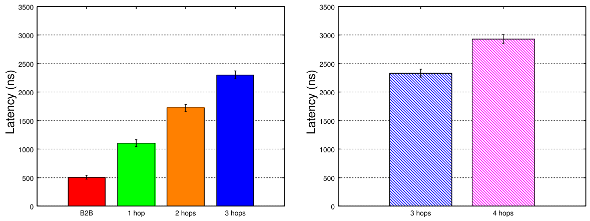


Fig. 10. Header format for FPGA experiments.



(a) (b)

Fig. 11. RDNA core switch latency: (a) no failure and (b) with failure.

small-form factor pluggable (SFP+) transceivers at 10Gbps line rate. Moreover, each board is configured to perform the steps described in Section II based on the received PRI or ERI value within each packet, with the corresponding defined switch pairwise prime values. As it is also shown in Figure 8(b), two different Anritsu traffic analyzers (MD1230B, MT1100A) were used to generate and monitor the traffic. The traffic analyzer performs the edge switch operation, i.e., ingress and egress on the programmable packet.

Figure 10 shows the extended high-level packet header format used in this experimental setup. Instead of using MAC address fields for embedding PRI and ERI, here we insert the route-IDs from the 31^{st} byte in order to make it evident that the RDNA architecture is actually protocol agnostic. In this case, it is necessary to wait for only 30 Bytes to be received (which takes 4 clock cycles in FPGA logic at 156.25Mhz, i.e., 25.6ns) to be able to read the route-IDs and to compute the modulo operation.

D. RDNA Low Latency and Fast Failure Reaction

In this setup, each FPGA has four 10Gbps Ethernet interfaces (see Figure 8b) and the used validation scenario is a full mesh topology, as shown in Figure 9. Due to the limitation on the number of FPGA cards, only the core network is considered in our testbed.

Figure 11(a) shows the latency measurements varying the number of hops $x = 1 \cdots 3$ in the core network. As can be seen, the latency for an RDNA Core switch is around 600ns per hop. To get the exact latency contribution of each hop, we have to subtract the 500ns from the total cumulative latency measured in the traffic analyzer loopback (B2B), i.e., no device under test with packet length of 1518 Bytes.

It is worth mentioning that the cumulative latency D at RDNA core may be computed as $D = (x \times 600 + 500)\,ns$, where x is the number of hops. An important observation is that the latency variability (jitter) is in order of tens of nanoseconds (ns), showing a potential packet-switched network with circuit switching like guarantees.

For the cases of *link failures*, depicted in Figure 9(b)(c), there are two different cases with 3 and 4 hops, respectively. Note that there is no link failure detection mechanism implemented internally in the FPGA. It would be necessary to use additional components (e.g., power meters) in order to detect the link failure. Instead, we have decided to emulate link failures with a special programmable packet. Along with PRI and ERI (previously shown in Figure 10), an additional 1 Byte field was used to carry the ID of the switch that will emulate the link failure. Upon receiving this special packet, the switch whose ID matches the switch-ID carried by the packet (e.g., 5 as in the example shown in Figure 9(b)) triggers the emergency route configuration by overwriting PRI with ERI in the received packet. In this case,

TABLE IV
TECHNICAL SPECIFICATION

Model	SFP+ 10G	Processor	Memory	OpenFlow
CORSA DP2100	Up to 32	Intel Core	16GB DDR3, 120GB HD	1.3
NetFPGA SUME	Up to 4	Virtex-7 FPGA	4GB DDR3 SODIMM	-

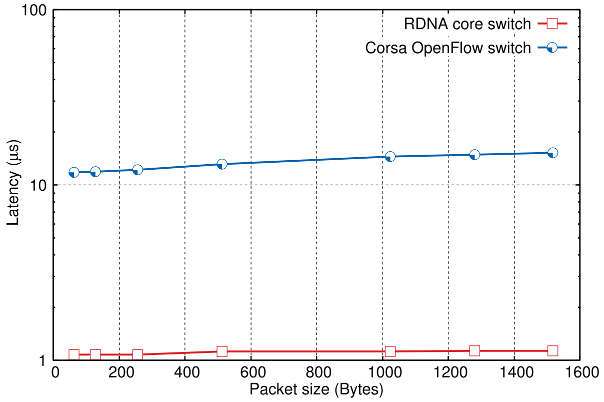


Fig. 12. Packet forwarding latencies for RDNA core switch and Corsa OpenFlow switch, varying the packet size for 1 hop.

the switch where the link failure is emulated introduces a delay of 25.6*ns* (4 clock cycles) to perform the overwriting process.

Figure 11(b) shows how fast is the failure reaction as it just consists of replacing PRI by ERI (25.6*ns*) and recomputing the modulo (6.4*ns*) to steer the packets to the emergency port. The latency is 2332*ns* for 3 hops and 2932*ns* for 4 hops. An important remark is that for the latency measured within the RDNA core we do not consider the failure detection time contribution to the latency, thus the measured latency is clearly a pure failure reaction time.

E. Performance of RDNA Switch Versus OpenFlow Switch

In order to compare the packet forwarding performance between RDNA and a traditional OpenFlow switch, we evaluated the RDNA core switch implemented on NetFPGA SUME versus a high performance OpenFlow switch designed by Corsa.[3] Corsas DP2100 is a SDN switching and routing platform that has 100G of non-blocking throughput, with hardware architected with Programmable Processing Units, powerful search engines, DDR3 memory, and is considered a first class ASIC based fabric. Table IV briefly describes the specifications of the switches used in this evaluation.

For this scenario, our benchmark follows RFC 2544 [37] that suggests the latency measures varying the Ethernet frame sizes (64, 128, 256, 512, 1024, 1280, 1518). RDNA Switch S_5 was selected as a device under test, while for Corsa's switch a simple match-action entry was added, i.e., all packets that arrive in the input port are forwarded to the output port with the lowest latency serving as a lower bound test.

Figure 12 shows the comparison results. For RDNA, the forwarding latency was around 1μs regardless of packet sizes, whereas for Corsa's OpenFlow switch, the latency was one order of magnitude greater than RDNA, achieving 11μs rising to 15μs (36%) for packet size of 64 and 1518 Bytes, respectively.

In regard to the differences on latency variability, the cut-through design on RDNA switch explains its characteristics towards a potentially circuit-like switching approach. For the Corsa switch, however, it relies on store-and-forward approach, therefore, the packet size has an impact on packet forwarding time, which means increase in latency by 36% even in the latency lower bound scenario. Although this approach is common for OpenFlow switches, which must wait for the identification of the flow [38], this is not effective in terms of latency guarantees in the forwarding mechanism.

V. RELATED WORK

This section reviews related work in three different areas: packet forwarding based on labels, resilient routing, and multicast communication.

A. Packet Forwarding Based on Labels

In literature there have been numerous proposals that make packet forwarding based on labels including Multiprotocol Label Switching (MPLS) [9], Virtual Local Area Network (VLAN), KeyFlow [16], and SDN-based architectures [10]–[12]. MPLS is a well-known source-routing protocol that forwards packets by writing and matching on labels attached to packets. Unlike RDNA that is tableless in the core, MPLS tags instruct the packet to travel hop-by-hop along a label-switched path with Label Distribution Protocol (LDP) forwarding tables.

Path Switching [11] proposes an alternative to MPLS for source routing which has the advantage of encoding forwarding information in a fixed amount of existing space in packet headers. However, there is only a high-level proof-of-concept with no experimental validation or testbed deployment so that it lacks evidence of its viability.

Segment Routing (SR) [39] is a proposal in which packets can be forwarded using SR forwarding tables and segment-IDs attached to packets. An ordered list of segments is encoded as a stack of labels. For packets processing, SR requires to rewrite the segment-ID using pop operations per SR node (top of the stack is considered the active segment-ID), whereas MPLS performs label swaps where the tag is swapped out for a new tag. In contrast, RDNA computes a simple modulo operation without swapping or pop operations per node in the core. RDNA allows to steer a flow through any path and service chain while maintaining per-flow state only at the edge nodes.

Although there have been other works that use CRT [18], [40] for proposing new routing schemes, our previous work KeyFlow [16] introduced a fabric-based model for core network architectures. RDNA, though, has driven its attention to EDC in which topologies are indubitably different. For instance, common EDC network topologies have two tiers (leaf and spine), where there are multiple paths available with the same length. Moreover, KeyFlow does not

[3] https://www.corsa.com/products/dp2100/

TABLE V
RECOVERY TIME WITH PROTECTION MECHANISMS [42]

	Network Convergence			
Protocol	**>250 ms**	**Sub 250 ms**	**50 - 150 ms**	**≈550μs**
STP (802.1D)	X			
RSTP (802.1w)	X			
MSTP (802.1s)	X			
RPVST+	X			
EtherChannel (LACP 802.3ad)		X		
Flex Links			X	
MPLS Fast Reroute [41]			X	
RDNA				X

consider hardware implementations, failure recovery issues and multicast communications.

B. Resilient Routing

A second bunch of work has been dedicated to fast failure reaction in order to provide resilient routing. In table V, we present an analysis of failure recovery time including a list of protocols that provide protection mechanisms. This shows a guidance for the resiliency protocol guarantees to meet the application protection requirements. One of the reasons for recovery times being greater than 50*ms* is the long time to update distributed tables. In the case of MPLS Fast Reroute [41], it still requires the support of a signaling protocol such as LDP for MPLS enabled switches. Thanks to the simplicity of replacing PRI by ERI triggered only by the switch where the failure is detected, RDNA allows ultra-fast failure recovery without table updates or additional control messages.

Comparing to our previous work KAR [13], the difference is on the deflection guided mechanism to drive packets to their destination under the presence of link failure. However, to define a full protection path along the entire route, the length of the bits required to support guided deflections may increase considerably. RDNA has extended it to support deterministic routing with programmable protection, but also differs fundamentally on the alternative paths encoding. Rather than using guided deflections, RDNA relies on two segments (route-IDs) allowing to protect the entire route. Also, as suggested by our FPGA prototype implementation, RDNA is easily supported in hardware-based devices. There is no need for a new protocol as it may be implemented by reusing the existing header fields to compactly encode the path attached to packets.

C. Multicast Communication

Multicast has been broadly studied in the context of wide-area networks [43]. EDC, however, differs in many ways from the wide-area profile given that a single administrative domain has control over the entire topology and it is no longer needed to run the decentralized protocols like IGMP and PIM.

RDNA multicast is not the first system to build the multicast tree based on a specific encoding scheme. Previous works [44]–[46] have encoded link identifiers inside packets using bloom filters. BIER [47] encodes group members as bit strings. However, these approaches are complex to implement and they are not able to process multicast traffic at line rate [23].

In Elmo [23], despite its design operating at line rate by means of modern programmable data planes, network states are required at intermediary switches tables. Also, it depends on control plane notification to recover from network failures.

RDNA differs from all proposals available in the literature, relying on polynomial function orchestrated by the CRT [18], which allows us to reduce the overhead to represent a multicast tree at 2-tier Clos network topologies. Hence, RDNA takes advantage of the topology characteristics to encode its residues forwarding graphs and actions to be performed by edge switches. Our analysis shows that a 200-byte header is sufficient to support huge multicast groups. Furthermore, RDNA is inexpensive to implement on modern programmable switches and supports the EDC requirements.

VI. CONCLUSION

This work proposed and experimentally proved the principles of RDNA: a tableless, protected source-routed traffic engineering capable, and topology-independent solution for ultra-reliable low-latency EDC with native multicast functionality. An extensive scalability analysis investigated 23 types of 2-tier Clos network benchmarking RDNA with existing approaches. Our evaluation quantifies the RDNA encoding for unicast, resilient routing and multicast communication, considering micro datacenters topologies as a reference design. Experimental results show that RDNA achieves *low-latency (600 ns)* in the core at 10 Gbps and virtually no jitter.

In an RDNA domain, a programmable network configuration and its policy can be seamlessly applied; for instance, service connectivity protection as policy is expressed by the RDNA controller that instructs the edge nodes installing the flows with their actions. For this case, our prototype demonstrated that RDNA is able to offer carrier grade protection achieving *ultra-reliable communication* with sub-milliseconds for failure recovery.

As future works, we intend to investigate operational factors of RDNA, such as power consumption aspects as in [7] and then devote efforts to enable slicing to be supported for multi-tenant solutions over EDCs. Another important enabler for 5G networks over RDNA to be implemented is Network Functions Virtualization (NFV) [48]. Evolving RDNA to accommodate convergent networks, i.e., integration fiber-wireless tailored to EDC infrastructures [49], is yet to be exploited.

REFERENCES

[1] J. Rexford and C. Dovrolis, "Future Internet architecture: Clean-slate versus evolutionary research," *Commun. ACM*, vol. 53, no. 9, pp. 36–40, Nov. 2010.

[2] "IMT vision framework and overall objectives of the future development of IMT for 2020 and beyond," Int. Telecommun. Union, Geneva, Switzerland, ITU-Recommendation M.2083-0, 2015.

[3] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, 2017, pp. 47–54.

[4] G. Araniti, M. Condoluci, P. Scopelliti, A. Molinaro, and A. Iera, "Multicasting over emerging 5G networks: Challenges and perspectives," *IEEE Netw.*, vol. 31, no. 2, pp. 80–89, Mar./Apr. 2017.

[5] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving SDN," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2012, pp. 85–90.

[6] C. Trois, M. D. D. Fabro, L. C. E. de Bona, and M. Martinello, "A survey on SDN programming languages: Toward a taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2687–2712, 4th Quart., 2016.

[7] S. S. Cercós, "Tackling OpenFlow power hog in core networks with KeyFlow," in *Electron. Lett.*, vol. 50, no. 11, pp. 1847–1849, Nov. 2014.

[8] H. L. Garner, "The residue number system," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 2, pp. 140–147, Jun. 1959.

[9] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," IETF, Fremont, CA, USA, Rep. RFC 3031, 2001.

[10] R. M. Ramos, M. Martinello, and C. E. Rothenberg, "SlickFlow: Resilient source routing in data center networks unlocked by OpenFlow," in *Proc. IEEE Conf. Local Comput. Netw.*, 2013, pp. 1–8.

[11] A. Hari, T. V. Lakshman, and G. Wilfong, "Path switching: Reduced-state flow handling in SDN using path information," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2015, pp. 1–36.

[12] R. MacDavid *et al.*, "Concise encoding of flow attributes in SDN switches," in *Proc. Symp. SDN Res. (SOSR)*, 2017, pp. 48–60.

[13] R. R. Gomes, A. B. Liberato, C. K. Dominicini, M. R. N. Ribeiro, and M. Martinello, "KAR: Key-for-any-route, a resilient routing system," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. Workshop (DSN-W)*, Jun. 2016, pp. 120–127.

[14] M. Martinello *et al.*, "Programmable residues defined networks for edge data centres," in *Proc. 13th Int. Conf. Netw. Service Manag. (CNSM)*, 2017, pp. 1–9.

[15] W.-K. Jia, "A scalable multicast source routing architecture for data center networks," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 116–123, Jan. 2014.

[16] M. Martinello, M. Ribeiro, R. de Oliveira, and R. de Angelis Vitoi, "KeyFlow: A prototype for evolving SDN toward core network fabrics," *IEEE Netw.*, vol. 28, no. 2, pp. 12–19, Mar./Apr. 2014.

[17] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM (SIGCOMM)*, 2014, pp. 503–514.

[18] C. Ding, D. Pei, and A. Salomaa, *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. River Edge, NJ, USA: World Sci., 1996.

[19] J. Hill. (Feb. 2014). *Complexity of the (Effective) Chinese Remainder Theorem*. [Online]. Available: http://www.untruth.org/josh/math/effective-crt.pdf

[20] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 159–170, Aug. 2006.

[21] J. Moy, "Multicast extensions to OSPF," IETF, Fremont, CA, USA, RFC 1584, Mar. 1994. [Online]. Available: https://tools.ietf.org/html/rfc1584

[22] X. Li and M. J. Freedman, "Scaling IP multicast on datacenter topologies," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2013, pp. 61–72.

[23] M. Shahbaz *et al.*, "Elmo: Source-routed multicast for cloud services," *CoRR*, vol. abs/1802.09815, Feb. 2018. [Online]. Available: http://arxiv.org/abs/1802.09815

[24] J. Oueis, E. Calvanese-Strinati, A. D. Domenico, and S. Barbarossa, "On the impact of backhaul network on distributed cloud computing," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, 2014, pp. 12–17.

[25] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[26] K. He *et al.*, "Presto: Edge-based load balancing for fast datacenter networks," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 465–478.

[27] M. Girolimetto, M. H. M. Paiva, R. S. Tessinari, C. Pavan, and F. O. Lima, "Two deterministic strategies for finding shortest pairs of edge-disjoint paths," in *Proc. Spring School Netw. Redes Para Ciudades Inteligentes (SSN)*, 2017, pp. 1–6.

[28] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[29] B. Martini, F. Paganelli, P. Cappanera, S. Turchi, and P. Castoldi, "Latency-aware composition of virtual functions in 5G," in *Proc. NetSoft*, 2015, pp. 1–6.

[30] RYU Physical Therapy. (Feb. 2014). *RYU SDN Framework Using OpenFlow 1.3*. [Online]. Available: http://osrg.github.io/ryu-book/en/Ryubook.pdf

[31] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, 2010, Art. no. 19.

[32] B. Pfaff *et al.*, "Extending networking into the virtualization layer" in *Proc. 8th ACM Workshop Hot Topics Netw.*, 2009, p. 6.

[33] J. Oostenbrink, N. L. M. van Adrichem, and F. A. Kuipers, "Fast failover of multicast sessions in software-defined networks," *CoRR*, vol. abs/1701.08182, pp. 1–15, Jan. 2017.

[34] N. L. M. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw. (EWSDN)*, 2014, pp. 61–66.

[35] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2013, pp. 43–48.

[36] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62–76, Feb. 1993.

[37] S. Bradner and J. McQuaid, "Benchmarking methodology for network interconnect devices," IETF, Fremont, CA, USA, RFC 2544, 1999.

[38] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella, "Simultaneously reducing latency and power consumption in OpenFlow switches," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 1007–1020, Jun. 2014.

[39] E. C. Filsfils *et al.*, "Segment routing architecture," IETF, Fremont, CA, USA, Netw. Working Group, Internet-Draft Segment Routing Architecture draft-ietf-spring-segment-routing-09, 2014. [Online]. Available: https://tools.ietf.org/html/draft-ietf-spring-segment-routing-09

[40] Y. Ren, T.-T. Tsai, J.-C. Huang, C.-W. Wu, and Y.-C. Tseng, "FlowTable-free routing for data center networks: A software-defined approach," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–6.

[41] G. Swallow, P. Pan, and A. Atlas, "RSVP-TE fast reroute," IETF, Fremont, CA, USA, RFC 4090, May 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4090.txt

[42] *Deploying the Resilient Ethernet Protocol (REP) in a Converged Plantwide Ethernet System (CPWE) Design Guide*, Cisco, San Jose, CA, USA, 2015.

[43] S. Islam, N. Muslim, and J. W. Atwood, "A survey on multicasting in software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 355–387, 1st Quart., 2018.

[44] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line speed publish/subscribe inter-networking," in *Proc. ACM SIGCOMM Conf. Data Commun. (SIGCOMM)*, 2009, pp. 195–206.

[45] S. Ratnasamy, A. Ermolinskiy, and S. Shenker, "Revisiting IP multicast," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2006, pp. 15–26.

[46] D. Li, H. Cui, Y. Hu, Y. Xia, and X. Wang, "Scalable data center multicast using multi-class bloom filter," in *Proc. 19th IEEE Int. Conf. Netw. Protocols*, 2011, pp. 266–275.

[47] A. Giorgetti *et al.*, "Bit index explicit replication (BIER) multicasting in transport networks," in *Proc. Int. Conf. Opt. Netw. Design Model. (ONDM)*, May 2017, pp. 1–5.

[48] C. K. Dominicini *et al.*, "VirtPhy: Fully programmable NFV orchestration architecture for edge data centers," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 817–830, Dec. 2017.

[49] M. Channegowda, R. Nejabati, and D. Simeonidou, "Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations [invited]," *J. Opt. Commun. Netw.*, vol. 5, no. 10, pp. A274–A282, Oct. 2013.

Alextian Liberato received the Ph.D. degree from the Universidade Federal do Espírito Santo in 2018. He is a Lecturer with the Department of Informatics, Federal Institute of Espírito Santo. His research interests include network architecture, datacenter designs, routing, and SDN.

Magnos Martinello received the Ph.D. degree from the Institut National Politechnique de Toulouse, France, in 2005. In 2008, he joined the Informatics Department, Federal University of Espírito Santo. He was a Visiting Researcher with the High Performance Network Group, University of Bristol from 2016 to 2017. His research interests include SDN, NFV, cloud computing, and network performance analysis.

Roberta L. Gomes received the Ph.D. degree from Université Paul Sabatier, France, in 2006. Her research interests include collaborative systems, distributed systems, and smart city platforms and infrastructures.

Arash F. Beldachi received the Ph.D. degree in dynamically reconfigurable network-on-chip from the University of Bristol, where he is a Senior Research Associate of high-speed FPGA and embedded systems design with High Performance Networks Group. His research areas of interest are reconfigurable systems, network-on-chip, system-on-chip, energy efficient digital systems, hardware software co design, and high-speed FPGA-based systems.

Emilio Salas received the B.Sc. degree in electronics and communications engineering from the Monterrey Institute of Technology and Higher Education, México, and the M.Sc. and Ph.D. degrees from the University of Essex, Colchester, U.K. His research areas of interest are optical communications and high performance networks.

Rodolfo Villaca received the Ph.D. degree in computer engineering from the University of Campinas in 2013. He is an Assistant Professor with the Industrial Technology Department, Federal University of Espírito Santo. His research interests are computing systems and networks.

Moisés R. N. Ribeiro received the Ph.D. degree from the University of Essex, U.K., in 2002. In 1995, he joined the Department of Electrical Engineering, Universidade Federal do Espírito Santo. He was a Visiting Professor with the Photonics and Networking Research Laboratory, Stanford University from 2010 to 2011. His research interests include fiber optic communication and sensor devices, systems, and networks.

Koteswararao Kondepu received the Ph.D. degree in computer science and engineering from the Institute for Advanced Studies Lucca (IMT), Italy, in 2012. His research interests include 5G, softwarization and virtualization of mobile networks, optical networks design, wired-wireless access convergence, communication networks reliability, energy efficient schemes in communication networks and sparse senor networks.

George Kanellos received the Ph.D. degree from the Photonics Communications Research Laboratory, National Technical University of Athens in 2008. He is currently a Lecturer with the High Performance Networks Group, University of Bristol. His research interests proposing the development of novel bio-photonic sensors.

Reza Nejabati (SM’02) received the Ph.D. degree in electronic systems engineering from the University of Essex. His current area of research is in the field of disruptive new Internet technologies with focus on application of high-speed network technologies. He is a fellow of the U.K. Higher Education Academy.

Alexander Gorodnik is a Professor of mathematics with the University of Bristol. His research interests are in the theory of dynamical systems and its connections with other branches of mathematics, such as number theory, geometry, and representation theory.

Dimitra Simeonidou is a Professor of high performance networks with the University of Bristol. He is a leading academic and entrepreneur in the fields of optical networks, cloud networking, and software defined networking. Her research is focusing in the fields of high performance networks, data centre networking, software defined transport networking, and smart cities.