Benefits of Programmable Topological Routing Policies in RINA-enabled Large-scale Datacenters

Sergio Leon, Jordi Perelló, Davide Careglio
Universitat Politècnica de Catalunya (UPC)
Barcelona (Spain)
Email: slgaixas@ac.upc.edu

Eduard Grasa
Fundació Privada i2CAT
Barcelona (Spain)
Email: eduard.grasa@i2cat.net

Diego R. López, Pedro A. Aranda
Telefónica I+D
Madrid (Spain)
Email: diego.r.lopez@telefonica.com

***Abstract*— With the proliferation of cloud computing and the expected requirements of future Internet of Things (IoT) and 5G network scenarios, more efficient and scalable Data Centers (DCs) will be required, offering very large pools of computational resources and storage capacity cost-effectively. Looking at todays' commercial DCs, they tend to rely on well-defined leaf-spine Data Center Network (DCN) topologies that not only offer low latency and high bisectional bandwidth, but also enhanced reliability against multiple failures. However, routing and forwarding solutions in such DCNs are typically based on IP, thus suffering from its limited routing scalability. In this work, we quantitatively evaluate the benefits that the Recursive InterNetwork Architecture (RINA) can bring into commercial DCNs. To this goal, we propose rule-based topological routing and forwarding policies tailored to the characteristics of publicly available Google's and Facebook's DCNs. These policies can be programmed in a RINA-enabled environment, enabling fast forwarding decisions in most scenarios with merely neighboring node information. Upon DCN failures, invalid forwarding rules are overwritten by exceptions. Numerical results show that the scalability of our proposal depends on the number of concurrent failures in the DCN rather than its size (e.g., number of nodes/links), dramatically reducing the total amount of routing and forwarding information to be stored at nodes. Furthermore, as routing information is only disseminated upon failures across the DCN, the associated communication cost of our proposals largely outperforms that of the traditional IP-based solutions.**

Keywords— Data center network; RINA; topological routing

I. INTRODUCTION

Looking for superior efficiency, uptime and scalability, nowadays' commercial Data Centers (DCs) tend to rely on well-defined leaf-spine Data Center Network (DCN) topologies that not only offer low latency and ultra-high bandwidth for server-to-server communications, but also enhanced reliability against multiple concurrent failures. Examples of this reliance are the Google's and Facebook's DCN topologies available in [1] and [2], respectively Moving toward future Internet of Things (IoT) and 5G network scenarios, a plethora of emerging innovative cloud services are expected to proliferate. This will put stress upon current DCs, requiring them to grow even larger in terms of computing resources. However, routing and forwarding solutions in DCNs, typically based on TCP/IP, do not scale well, resulting in large forwarding table (at least in the order of several tens of thousands of entries in highly-optimized configurations [3]), routing burden and communication cost (information exchanged to populate routing tables and re-converge upon failures). This problem was identified longtime ago, but the TCP/IP protocol suite, not designed for cloud networking, limited the improvements that achievable by the solutions proposed in the literature [4].

In contrast to the rigidness of the TCP/IP protocol stack, the clean-slate Recursive Internetwork Architecture (RINA) [5] brings a programmable environment [6] where Quality of Service (QoS), security, routing and forwarding policies in forwarding devices can be fully configured by the network administrator. This opens the door to the deployment of policies tightly tailored to the specific DCN characteristics inside a RINA-enabled DC, outperforming solutions based on TCP/IP, whose protocols were optimized for the delivery of a best-effort Internet with an arbitrary topology, a very different environment to that of a DCN.

This work aims to quantify the benefits that topological routing and forwarding policies can offer in a RINA-enabled large-scale DCN. Our policies make use of the DCN topology knowledge to forward packets to the closest neighboring device to their destination based on rules. In the non-failure scenario, this approach only requires the storage of forwarding information per adjacent neighbor (compared to traditional forwarding tables, which may contain up to one entry per network node). Upon failures in the DCN, some forwarding rules may not succeed to deliver packets to destination. In this case, few exceptions overriding those rules are stored at forwarding devices, the only time when additional forwarding information is required.

The remainder of this paper continues as follows. Routing solutions for DCNs available in the literature are reviewed in section II. Next, in section III we introduce the assumed RINA-enabled DC scenario based on Google's and Facebook's DCNs characteristics. The configuration of the routing and forwarding policies in both use cases are further elaborated in Sections VI and V. In Section VI, we provide numerical results comparing our forwarding and routing

This work is partly funded by the European Commission through the FP7 PRISTINE project (FP7-619305). Moreover, it has been partly funded by the Spanish project SUNSET (TEC2014-59583) that receives funding from FEDER.

policies against current solutions based on TCP/IP. Finally, Section VII draws up some conclusions.

II. Related Work

Given the regular and known topology of a DCN, deterministic routing [7] was the scheme initially deployed in many DCs. In such a scheme, the addresses of nodes are based on their topological properties, so that the route between any pair of nodes is known beforehand and does not change over time. The route is usually encoded in the packets in the form of a bit-stream or coordinates (e.g., see [8]). While scalable, this rigid scheme has two major drawbacks: the lack of automation in defining addresses, and thus the setup of routes, and no multipath support, preventing the recovery upon DCN failures. The valiant routing scheme [9] was proposed as a solution to overcome such deterministic routing shortcomings, bringing multipath support and load balancing. For a communication between any pair of nodes, a random intermediate address *i* is selected first and the path is composed by routing packets from source to *i* and then from *i* to destination. This way, multipath support is enabled, but at expenses of longer paths and still an automation process for the naming.

The adopted routing scheme in many large DCs is today based on IP because of the low-cost of IP-based commodity servers. To mitigate the inherent limitations of routing solutions initially designed for an Internet with arbitrary topology, modifications to link-state and path-vector routing have been introduced. For example, Facebook's DCN uses BGP-4 [10] to avoid the need for an address per interface (as required by IP), assigning an ASN per node, routing to the node instead of to the interface [11]. Nonetheless, BGP-4 suffers from many limitations, e.g., path exploration upon failures, manual configuration of timers, TCP connections between any pair of connected ASNs, etc. As a result, these schemes imply a high communication cost and require many entries in routing and forwarding tables to take optimal routing decisions and allow route recovery upon failures.

A new trend in intra-DC routing is a Software Defined Networking (SDN) approach centralizing all forwarding decisions, where only a few nodes know the state of the full DCN. For example, Google's DCN uses its SDN-based approach to control packet forwarding within the DCN [1]. Although this strategy allows taking efficient decisions at low communication cost, the complexity of centralized decisions increases with the network size, potentially imposing scalability issues as the network grows larger.

III. Scenario under study

RINA is a computer network architecture that unifies distributed computing and telecommunications [5]. RINA's fundamental principle is that computer networking is Inter-Process Communication (IPC). RINA reconstructs the overall structure of the Internet forming a model that comprises a single repeating layer, the DIF (Distributed IPC Facility). Each DIF instance implements the same functions and mechanisms, which are configured via policies in order to adapt to the specific scope (operating environment). In this paper we focus on a RINA deployment inside a DC following the DIF setup depicted in Fig. 1. Such a RINA-enabled DCN network is partitioned into three main types of DIFs of different scopes: i) a single DC-Fabric DIF, acting as a large distributed switch; ii) a DC DIF that connects all servers in the DC together under the same pool; and iii) multiple tenant DIFs, isolated and customized as per the requirements of the different tenants. Note that in the figure the underlying point-to-point links are abstracted as "shim" DIFs, which allow the deployment of RINA over legacy technologies or physical media [12].

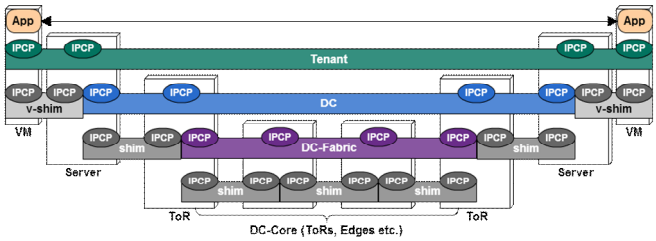

Fig. 1. DIF setup inside a DC between Virtual Machines (VMs) running in DC servers. The DC-Fabric DIF (violet color) is the focus of this work.

In this work, we focus on the DC-Fabric DIF, that is, the one providing connectivity between Top-of-the-Rack (ToR) switches and between edge routers and ToR switches. To provide outcomes applicable to realistic DC scenarios, we assume that the DC-Fabric DIF follows the topologies of the large Google's and Facebook's DCNs shown in Fig. 2. As for the Google's DCN topology (Fig. 2, top), a unique plane of spine switches interconnects all pods and edge planes in the DCN, offering multiple equal cost paths between each pair of ToRs and edges, even under multi-failure scenarios. Regarding the Facebook's DCN (Fig. 2, bottom), the fabric switches of a pod connect each one to a distinct spine set that provides connectivity to all other pods and to edge nodes. Again, high redundancy is introduced to survive multiple concurrent failures across the DCN.

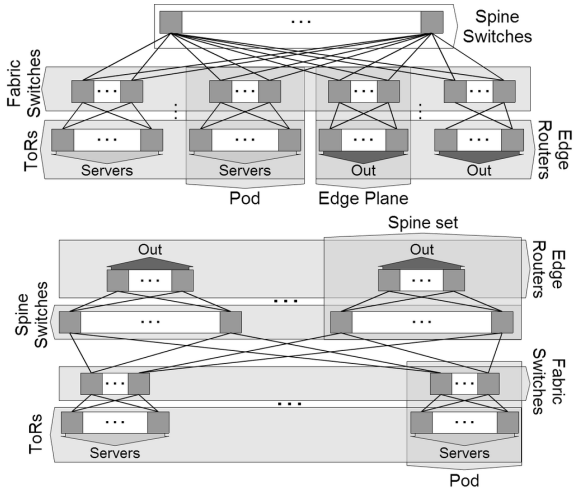

Fig. 2. Google's (top) and Facebook's (bottom) DCN topologies, extracted from references [1] and [2].

A key benefit of RINA is its programmable behavior using policies, as well as the automation of the enrollment and naming processes of a new node in a DIF [6]. This allows to get rid of the constraints imposed by both the conventional deterministic routing and the IP-based solutions, opening an opportunity for cheap customizable hardware. In the following section we elaborate on topological routing and forwarding

policies particularly designed for a potential DC-Fabric DIF in the Google's DC premises, exploiting its DCN characteristics for superior efficiency and scalability.

IV. RINA-ENABLED GOOGLE'S DCN USE CASE

Taking a look at Google's DCN, we can see that the DC-Fabric DIF can be described by only 6 parameters: Number of spine nodes (s), number of pods (P), number of edge planes (E), number of fabric nodes per pod/edge plane (f), number of ToRs per pod (t) and number of edges per edge plane (e). Moreover, given the regularity of the topology, we find that only a few types of nodes exist. This becomes particularly useful for performing topological forwarding, which depends on the relations between node locations. Specifically, we have spine switches (hereafter referred as R0 nodes), fabric switches (R1 nodes) and ToR switches/edge routers (R2 nodes).

From now on, we are going to refer to pods and planes (of spine or edge switches) indistinctly as groups. Taking advantage of R0, R1 and R2 node groups in the DCN, we propose a location-dependent but route-independent node-addressing scheme A.B, where A identifies a group and B is the identifier of the node within the group. Although simple, this scheme allows inserting the topological location of any node in the DCN in its address as follows:

A = 0, B ∈ [1, s] → Spine B
A ∈ [1, P+E], B ∈ [1, f] → Fabric B at pod/Edge Plane A
A ∈ [1, P], B ∈ (f, f+t] → ToR B at pod A
A ∈ (P, P+E], B ∈ (f, f+e] → Edge B at edge Plane A

A. Forwarding policy

A key requirement of any forwarding policy is the ability to quickly decide the neighboring node to which a packet must be forwarded to. Forwarding policies in RINA are not restricted to be a traditional table. Instead, any forwarding function capable to quickly perform accurate forwarding decisions can be used. To this avail, we leverage on the regular topology of Google's DCN (Fig. 2, top) to design a minimalistic forwarding function.

Being aware of the specific DCN topology (from the set of parameters listed at the beginning of this section) and the location of the node in it, only forwarding entries to adjacent neighbors need to be stored at DCN nodes and simple forwarding rules can be used (detailed in the next sub-section). When failures occur across the DCN, however, it may happen that primary forwarding rules fail in delivering a packet to its destination. Thus, we require exceptions to overwrite the erroneous decisions of primary rules. These exceptions are similar to traditional forwarding table entries, but are only required upon certain failure scenarios. Moreover, the total number of exceptions tends to be considerably smaller than the number of entries required in a traditional forwarding table (at most the same in the very worst case), as many communications across the DCN remain unaffected by specific link or node failures. Therefore, only storing exceptions to primary rules upon failures can yield a large reduction in terms of memory usage compared to a traditional forwarding table.

1) Forwarding rules

In order to quickly access neighbor information, we assign a locally unique identifier (Neighbor-Id) to every neighbor, abstracting its real address. By using Neighbor-Ids as index, we can store all neighbor nodes' information, including port address or status, in a direct access structure. These Neighbor-Ids are assigned as follows:

At **R0**: R1 → (A - 1) * f + B - 1
At **R1**: R0 → B - 1, R2 → s + B - f – 1
At **R2:** R1 → B - 1

Forwarding rules use Neighbor-Ids to easily define the set of valid neighbors to reach any destination across the DCN. Given the nature of the communications inside a DC (over the DC-Fabric DIF in a RINA-enabled scenario), only end-to-end flows between R2 nodes (ToR switches and edge routers) will be established. Therefore, forwarding rules only need to consider R2 nodes as possible destinations. These rules are depicted in Fig. 3.

At **R0: Rule (A.B)** → [(A – 1) * f, A * f)
At **R1: Rule (A.B)** → [0, s)
At **R2: Rule (A.B)** → [0, f)

Fig. 3. Pseudo-code of primary forwarding rules in the Google's DCN

Let us show how those rules work to reach any R2 node A.B. At R0 nodes, any neighbor R1 node of the group A can be used to reach the destination (an ECMP-like policy can be chosen to load-balance the traffic). In an R1 node, we have two possibilities: either we are in a different group than the destination, so that we can use any R0 node to reach that (again load-balancing can be used), or we are in the same group (A), never reaching the rule as it is a direct neighbor. Finally, at R2 nodes we can use any R1 neighbor to reach any other R2 in the network.

In all cases we have a range of valid neighbors. Then, upon having an unreachable neighbor, the rules would simply remove it from the valid ones. This allows keeping most of the rules still valid when failures affect the current node (otherwise multiple exceptions could be necessary).

2) Forwarding exceptions

Focusing on the exceptions to reach R2 nodes, we find 3 kinds of them: to a specific node A.B, to a specific group A and to all other groups. It should be noted that exceptions to other groups are used neither in R0 nodes nor in R1 or R2 nodes for destinations in the same group.

Given the high number of neighbors that some nodes have, an important point is how Neighbor-Ids are stored at the exception entries. When encoding exceptions, we use two different encoding modes, being the use of one or another specified as a flag in the exception header. With the default encoding, the stored Neighbor-Ids represent the valid neighbors to reach the destination. When the number of valid neighbors is high, an inverse encoding can alternatively be used, where the stored Neighbor-Ids represent the invalid neighbors to reach a destination. Jointly with inverse encoding at R1 nodes, a direction flag is used to specify if that list applies to only R0 nodes (UP), R2 nodes (DOWN) or both. This proposal yields significant memory optimization, as most exceptions can be described as "To reach X go UP/DOWN, without using Y".

3) Forwarding decision

Putting together direct routes to neighbors, exceptions and rules, the full forwarding decision can be described by the simple pseudo-code in Fig.4.

```
Forward (A.B)
If is Connected Neighbor (A.B) → Forward (A.B)
If A = 0 || B <= f  → Unreachable
If is Exception (A.B)  → Exception (A.B)
If is Exception (A)  → Exception (A)
If (My A != A & is Exception ())  → Exception ()
Else  → Rule (A.B)
```

Fig. 4. Forwarding pseudo-code with exceptions and primary rules

The forwarding function needs to be executed per packet. DCN performance requirement will most probably impose forwarding rules implementation to be on hardware. For this, we first have that neighbors can be stored in a direct access structure, making these last hops automatic. Then, taking profit from the small number of exceptions, we can have them ordered as R2 nodes, specific groups or other groups and simply iterate them until finding the first match. Being desirable for flows to maintain the same path during its lifetime (to prevent packet reordering), both rule and exception execution can use a fast hashing of the flow identifier of the packet to decide on the next hop, instead of deciding it randomly.

B. Routing policy

The previously described forwarding policy requires knowing the affected routes to destinations upon failures and how to alternatively reach them. Hence, the routing policy has to provide enough information to populate such exceptions. While a simple link-state or distance-vector routing protocol could be used to obtain exceptions to the primary rules upon failure scenarios, we can compute them more efficiently by exploiting the complete DCN topology knowledge that nodes have. Indeed, there is no need for nodes to propagate the state of operational resources across the network, but only that of those experiencing failures. To this end, we propose the link-failure routing policy, a variation of link-state routing based on failure propagation, where instead of having all nodes propagating their full neighbor table, only failed links are propagated while the rest is assumed to be working, resulting in a large reduction of the information exchanged and stored at network nodes.

Although we could use the DCN topology and failed links' knowledge to compute the forwarding exceptions using a Dijkstra's routing algorithm, such an approach has a significant computational cost and does not scale well. Instead, we found that with a list of failures we could restrict our search to problematic locations and compute the exceptions directly, if some constraints on valid paths are considered. Constraints on valid paths are, in fact, required to reduce the complexity of the algorithms. Even so, those are thought taking into account the high number of available paths towards any R2 node, and that it is better to have unreachable destinations (with possible movements of VMs) than filling the network with traffic routed through sub-optimal paths. For example, we consider the following two constraints at R0 nodes: 1) a group is reachable if it has at least one R1 neighbor connected to at least one R2; 2) an R2 node is reachable if it has at least one R1 neighbor for which there exists a 1 or 3 hops path to reach it in the group.

C. Computing the forwarding exceptions

Given the list of possible failures in the DC-Fabric DIF, we parse and process them in order to compute the exceptions to problematic destinations. For example, for R0 nodes, the pseudo-code in Fig. 5 can be used.

```
Parsed data and functions used:
unreachableGroups ←groups with all R1 unreachable
unreachableNodes ←R2 disconnected from all R1
R2Fails ←R2 disconnected from some R1
R1notReachR2 ←R1 with problems reaching R2
Reachable (A.B) ← Check if neighbor A.B can be reached
reachableGroupR1(A) ← reachable A.* R1s
reachableR1At (A.B) ← reachable R1s from R2 (A.B)
reachableR2At(A.B) ← reachable R2s from R1 (A.B)
Algorithm:
Exceptions = Ø
if I am disconnected then return Exceptions
GroupsWithProblems = Ø
for each A.B in R1notReachR2 do
  if Reachable(A.B) then GroupsWithProblems .add(A)
for each (A) in GroupsWithProblems do
  validPorts = Ø
  for i = 1..f do
    if Reachable(A.i) and A.i ∉ R1notReachR2 then
      validPorts .add (A.i)
  if validPorts == Ø then unreachableGroups.add(A)
  else Exceptions.add(A, validPorts )
for each (A) in unreachableGroups do E.add(A.0, Ø)
for each (A.B) in unreachableNodes do
  if (A) ∉ unreachableGroups then Exceptions.add(A.B, Ø)
for each A.B in R2Fails do
  if A ∈ unreachableGroups then continue
  if (A.B) ∈ unreachableNodes then continue
  myReach = reachableGroupR1(A)
  dstReach = reachableGroupR1At(A.B)
  if myReach ⊆ dstReach then continue
  if myReach ∩ dstReach != Ø then
    Exceptions.add(A.B, myReach ∩ dstReach)
    continue
  reachDst = Ø
  for each node (A.B') in dstReach do
    reachDst .add(reachableR2At (A.B'))
  reachNei = Ø
  for each node (A.B') in myReach do
    reachNei .add(reachableR2At(A.B'))
  validPorts = Ø
  if reachNei ∩ reachDst != Ø then
    validPorts .add(A.B')
  Exceptions.add(A.B, validPorts )
return Exceptions
```

Fig. 5. Pseudo-code for computing exceptions

Algorithms to compute exceptions like the one described in Fig. 5 aim to direct the search toward failures that may

require an exception, while discarding the rest. An example can be seen when failures between R0 and R1 nodes are only considered for the current node, as other ones are not included in feasible paths given the imposed constraints. Another case is seen for failures between R1 and R2 nodes where the depth of the search depends on the specific failures within the group, avoiding for example a search in depth if there are some shared R1 nodes between the current node and the one affected by failures. While such algorithms are fully dependent on the topology and require some constraints, they yield a significant improvement both in time and memory usage against the traditional route computation, as we do not need to compute and store reachability information to all destinations in the network, but only to problematic ones.

V. RINA-ENABLED FACEBOOK'S DCN USE CASE

Looking at the Facebook's DCN in Fig. 2, we observe that it can be described by only 5 parameters: Number of pods (P), number of fabric nodes per pod and spine sets (f), number of ToRs per pod (t), number of spine nodes per spine set (s) and number of edges per spine set (e). Although this DCN can be described by fewer parameters than the Google's one, here we have 4 types of distinct nodes: ToRs, fabric switches, spine switches and edge routers. Hence, we propose the following addressing scheme based on A.B.C addresses:

ToR: 0.Pod-Id.Tor-Id
Fabric: 1.Pod-Id.Spine-set
Spine: 2.Spine-set.Spine-Id
Edge: 3.Spine-set.Edge-Id

Like for the Google's DCN, we also propose a forwarding policy based on rules and exceptions. In this case, Neighbor-Ids are defined as follows:

At **ToR**: Fabric → C
At **Fabric switch**: ToR → s + C, Spine → C
At **Spine switch:** Fabric → A, Edge → P + C
At **Edge router:** Spine → C

ToR: Rule (A.B.C)
If A = 3 → {B}, Else → [0, f)
Fabric: Rule (A.B.C)
If A = 3 & B != My B → [s, s+t), Else → [0, s)
Spine: Rule (A.B.C)
If A = 0 → {P + C}, Else → [0, P)
Edge: Rule (A.B.C) → [0, s)

Fig. 6. Pseudo-code of primary forwarding rules in the Facebook's DCN

Fig. 6 details the pseudo-code of the primary forwarding rules proposed for the Facebook's DCN. With information of the current failures, exceptions to overwrite those rules can be computed in a similar way as in the Google's DCN (not detailed here due to the lack of space). Note in this case that losing a fabric switch automatically multiplies the path length to reach its pod from some edge routers. Therefore, when computing the respective exceptions, we require either to have less restrictive constraints in these cases, thus incrementing the complexity of the algorithms to avoid unreachable areas across the DCN, or more restrictive ones, reducing complexity but at expenses of having unreachable areas.

VI. COMPARISON WITH CURRENT SOLUTIONS

Current routing and forwarding solutions for IP impose multiple limitations that the RINA architecture already solves. For example, for addressing DCN devices, we are not forced to use 4 or 16-byte addresses as imposed by IPv4 or IPv6, but can use scenario-specific addresses. Besides, public addresses of servers/VMs are not propagated with routing updates, only focusing on the smaller set of node addresses in the DCN. While the benefits of RINA are enough to contemplate its usage inside DCs, we also want to quantitatively evaluate the performance of the proposed routing and forwarding policies against that of currently available solutions for the same purposes.

Aiming to analyze the number and size of traditional Forwarding Table entries vs. Rules plus exceptions in our policies, we have considered two different DC-Fabric DIFs. The first one, named DIF-Go, reproduces the Google's DCN topology, whereas the second one, named DIF-FB, reproduces that in the Facebook's DCs. Table I depict the parametrization of both DIFs, taking the number of pods (P) as base parameter. The expressions to determine the rest of parameters (as a function of P) allow us to obtain similar configurations as those reported for the real DCNs.

TABLE I. DETAILS OF THE DCN-FABRIC DIFS

Pods (P)		P
ToRs per pod (t)		P/2
Fabric switches per pod/edge plane (f)		$Log_3(P)$
DIF-Go	Edge planes (E)	P/4
	Edge routers per edge plane (e)	P/2
	Spine switches (s)	P
DIF-FB	Edge routers per spine set (e)	$P^2/8f$
	Spine switches per spine set (s)	P/2
Total number of servers		**$P^2/2$**
Total number of edges		**$P^2/8$**

As a first objective, we compare the number of entries in a forwarding table against the number of neighbor entries plus exceptions for large-scale DCNs. For this, we fix in our first tests P=100, resulting in DCNs with 5000 ToR switches. Being our approach dependent of the number of concurrent failures across the DCN, we perform our tests for 0, 1, 2, 5 and 10 concurrent ones, being those either link or node failures (randomly chosen). We perform 50000 tests for each DIF and number of failures, averaging the obtained results.

TABLE II. AVG. ENTRIES VS. MAX ENTRIES (%) GIVEN N FAILURES

Method\Failures	**0**	**1**	**2**	**5**	**10**
Exceptions	0.21	0.22	0.23	0.24	0.27
DIF-Go FWT	11.6	11.9	12.3	13.2	14.8
DIF-FB FWT	11.4	12.1	12.8	14.8	18.1

In Table II we can see the relative number (in %) of required entries in each case against the total number of DCN nodes. With our policies (named as Exceptions in the table), we require at most one exception per failure (as expected), thus being most of the stored entries related to adjacent neighbors'. With traditional forwarding tables (FWT), we assume that ToR and edge addresses can be aggregated at pods and edge planes/spines. With this, the number of entries can be lowered by 11 to 18% compared to the trivial FWT solution where one entry per destination node is stored. While

this represents a noticeable reduction, it cannot approach by far the scalability of our proposals.

In addition to the number of entries, we are also interested in comparing the amount of forwarding data stored. For this purpose, we focus on the amount of stored ports in the forwarding entries and exceptions rather than on their encoding, so as to become independent from specific data structures. Fig. 7 shows the average number of entries and stored ports in DIF-Go and DIF-FB for different P sizes (number of pods in the DCN), in scenarios from 0 to 10 concurrent link/node failures. With forwarding tables, we also limit the number of stored ports per destination to 16, a common limit in ECMP implementations. In the figure, we can see how, the number of forwarding entries and their size grow steadily with P. In contrast, our proposed solution only needs to store adjacent neighbors' information plus exceptions, remaining the number of forwarding entries almost constant as the size of the DIFs grows up.

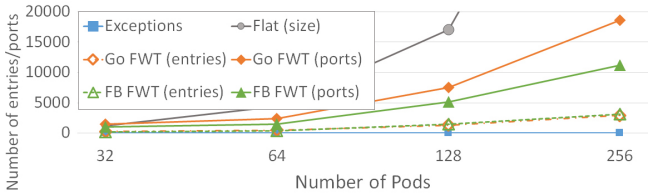

Fig. 7. Avg. number of entries and stored ports given the number of pods

Although the benefits of our proposals against the traditional forwarding tables are good enough to justify topology dependent policies, the benefits on the computational and communication cost of searching exceptions, given a specific number of failures, should also be investigated. Regarding the latter, we have a clear improvement in the sense that, as we know the topology, we can take some knowledge as granted. Given this knowledge of the topology, we can avoid the initial routing information flooding that any common on link-state or distance-vector routing protocol needs for populating nodes' routing and forwarding tables. Besides, most IP solutions require some type of refresh of routing information to ensure that the knowledge is updated. In our case, RINA DIFs can provide reliable communications between nodes, which can be configured in a DC-Fabric DIF as well. This makes routing information refreshes unnecessary.

Finally, in terms of computation cost, in order to validate our proposed approach to compute forwarding exceptions, we compare its complexity against a traditional solution based on link-state routing and Dijkstra's routing algorithm. For this purpose, we take the pseudo-code proposed for R0 nodes in the Google DCN in Fig. 5. Moreover, to simplify the results we consider the same parametrization described in Table I, and use the number of pods (P) and failures R as the two only parameters. With the traditional link-state solution, computing either exceptions or a forwarding table has a complexity lower bound of $o(P^2*Log(P))$. Conversely, with our approach we find a complexity upper bound of O(R*P). Note that this upper bound will never be reached, as it would require the same failures to affect R2 nodes in all groups. Since the number of concurrent failures in this type of networks is small by design, we found a lower upper bound, ensured for the cases where R < P/2. In these cases, if we use the known failures to check reachability between R1 nodes instead of a brute force approach, our complexity can be bounded to O(R*Log(R)* Log(P)), representing a big improvement in performance, still without considering that in the non-failure scenario we only have the constant cost of checking that there are no failures in the network.

VII. Conclusions

In this paper, we proposed rule-based topological routing and forwarding policies for RINA-enabled large-scale DCNs, based on those recently made publicly available by Google and Facebook. These policies use the knowledge of the DCN topological characteristics for superior efficiency and scalability, achieving fast and 100% successful forwarding decisions in the non-failure scenario with merely neighboring node information. Upon DCN link or node failures, forwarding exceptions are computed and stored at DCN nodes to override the possible invalid forwarding decisions of primary rules. To minimize the size of the stored exceptions, only the invalid neighbors (instead of all valid ones) are contemplated, something that gives a great improvement, given the large number of redundant paths to any destination. Regarding the proposed routing policy, only failed link information is disseminated, reducing the communication cost to a large extent. The obtained results in large-scale DCNs illustrate the perfect scalability of our routing and forwarding policies.

References

[1] Arjun Singh, et al., "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network". In SIGCOMM, London, United Kingdom, August 2015.

[2] Alexey Andreyev, "Introducing data center fabric, the next-generation Facebook data center network", available online at: https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/ .

[3] D. Arora, T. Benson, J. Rexford, "ProActive routing in scalable datacentres with PARIS". In DCC 2014, Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing.

[4] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, M. F. Zhani; "Data Center Network Virtualization: A survey".

[5] J. Day, I. Matta, and K. Mattar. "Networking is IPC: A Guiding Principleto a Better Internet". In CoNEXT'08: Proceedings of the 2008 ACMCoNEXT Conference, pages 1 6, New York, NY, USA, 2008. ACM.

[6] V. Maffione, F. Salvestrini, E. Grasa, et al. "A Software Development Kit to exploit RINA programmability". In IEEE ICC 2016, Kuala Lumpur, May 2016.

[7] M.E. Gómez, P. López, J. Duato. "A Memory-Effective Routing Strategy for Regular Interconnection Networks". In IPDPS'05 conference, 2005.

[8] S. Habib, F. S. Bokhari, and S. U. Khan, "Routing Techniques in Data Center Networks," in Handbook on Data Centers, S. U. Khan and A. Y. Zomaya, Eds., Springer-Verlag, New York, USA, 2015, ISBN 978-1-4939-2091-4, Chapter 16.

[9] K. Chen, et al. "Survey on routing in data centers: insights and future directions", IEEE Network, vol. 25, no. 4, pp. 6-10, July 2011.

[10] Y. Rekhter, T. Li, S. Hares. "A Border Gateway Protocol 4 (BGP-4)". RFC 4271, January 2006.

[11] P. Lapukhov, A. Premji, J. Mitchell. "Use of BGP for routing in large-scale data centers". IETF Network Working Group, draft-lapukhov-bgp-routing-large-dc-07, February 2014.

[12] S. Vrijders, E. Trouva, J. Day, E. Grasa, et al. "Unreliable IPC in Ethernet: migrating to RINA with the shim DIF". ICUMT 2013, Almaty.