Integrating Servers and Networking using an XOR-based Flat Routing Mechanism in 3-cube Server-centric Data Centers

Rafael Pasquini[1], Fábio L. Verdi[2] and Maurício F. Magalhães[1]

[1]Department of Computer Engineering and Industrial Automation (DCA)
School of Electrical and Computer Engineering (FEEC)
State University of Campinas (Unicamp)
C. P. 6.101 – 13.083-970 – Campinas – SP – Brazil

[2]Federal University of São Carlos (UFSCar)
18.052-780 – Sorocaba – SP – Brazil

{pasquini,mauricio}@dca.fee.unicamp.br

verdi@ufscar.br

Abstract. *This paper introduces the use of an XOR-based flat routing mechanism for server-centric data centers (DCs) organized in 3-cube topologies. Essentially, the networking DC architectures available in the literature create a black box network unaware about the servers, increasing the complexity to deploy services inside DCs. The main reason for this black box is the lack of scalability when the servers are inserted in the forwarding tables of the switches. The server-centric 3-cube topology used in this work directly connects servers, removing the need for switches and/or routers in order to forward traffic inside the DC, approximating the servers to the networking infrastructure. The proposed XOR-based flat routing mechanism introduces a new semantic for routing, where a small information about the entire network allows traffic forwarding across the DC, providing the required scalability for inserting the servers in the routing tables. This paper presents the proposed 3-cube server-centric DC networking architecture, the XOR-based flat routing mechanism, and evaluations demonstrating the achieved scalability in terms of signaling, routing table entries, stretch and load distribution.*

1. Introduction

Nowadays, innumerable applications demanding massive storage, memory and processing support are executed in large-scale DCs deployed by companies such as Microsoft, Amazon, Google and Yahoo. To cope with this scale and lower total ownership costs, these DCs use custom software such as BigTable [Chang et al. 2006], Google File System (GFS) [Ghemawat et al. 2003], Map-Reduce [Dean and Ghemawat 2008], and customized hardware encompassing servers [Cisco 2010, Google 2010], racks and power suppliers. However, the only component that has not really changed yet is the network, which usually leverages the current available Internet-related routing technologies.

As detailed in [Esteve et al. 2010], there is a set of requirements for the development of new DC architectures. For example, it is needed to consider how the DC communicates with external networks, such as the Internet, accepting external requests, homogeneously spreading these requests among the servers (normally using the Valiant

Load Balance (VLB) and/or Equal Cost Multi Path (ECMP)), and adopting hash-based mechanisms in order to preserve the communication sessions. In this way, there have been a number of proposals for redesigning DC networking architectures, many of them focusing on fat-tree topologies for scaling IP and/or Ethernet forwarding to large-scale DCs [Greenberg et al. 2008, Greenberg et al. 2009]. Nevertheless, the routing solutions used in these proposals are basically mapping services, in which the IPs and/or MAC addresses of servers are associated to the MAC addresses of ingress, intermediary and egress switches through the fat-tree topology. Consequently, this scenario achieves scalability by creating a black box totally unaware about the servers present in the DC, delegating the maintenance costs related to changes in the servers' structure to the mapping service. Basically, servers and network infrastructure maintain an oblivious relationship, increasing the complexity to deploy services inside the DCs.

This paper presents a server-centric DC architecture in which an XOR-based flat routing mechanism provides direct server-to-server communication at the layer 2, as an alternative to the scalability problems faced in other DC proposals. In the proposed scenario, a 3-cube topology is used to connect servers due to its intrinsic higher path redundancy, simplicity for wiring and fault resilience. The proposed approach, using the XOR-based flat routing mechanism in conjunction with the 3-cube topology, removes all the dependence on traditional switches and/or routers for server-to-server traffic forwarding inside the DC, closing the gap between servers and network infrastructure and, in this way, creating a server-aware DC for the deployment of services. Essentially, the proposed XOR-based flat routing mechanism, already instantiated in other scenarios [Pasquini et al. 2009, Pasquini et al. 2010a, Pasquini et al. 2010b], uses a metric based on the bitwise XOR operation to organize the routing tables in columns (called buckets). The main contribution of the proposed XOR-based mechanism is the capability of performing routing with a small knowledge about the entire network, providing the required scalability for integrating servers with the DC networking infrastructure.

Afterwards, the proposed XOR-based mechanism only requires uniqueness while assigning flat identifiers (IDs) to servers, being a totally random distribution of IDs ideal for its operation, i.e., no topological or semantical IDs organization is required. In this way, the IDs are assigned to serves inside the DC by simply selecting the smallest MAC address among the NICs present on each server, creating a plug-and-play environment for the configuration of new 3-cube-based DCs. Moreover, since it operates at layer 2, it transparently supports all the applications running on upper layers or inside virtual machines. According to it, this paper is focused on describing the proposed server-centric DC architecture, detailing the internal server-to-server communication using the XOR-based flat routing mechanism at the layer 2. Consequently, other architectural details, such as external communication with the Internet, are kept out of the scope of this paper.

The evaluations present in this work consider 3-cube topologies with 64, 128, 256, 512, 1024 and 2048 servers. All of them were evaluated using a developed emulation tool, where each server is executed as an independent thread, being able to exchange signaling messages to build the routing tables and forward traffic. The results detail the number of signaling messages, the number of entries in each routing table, the route stretch and the load distribution among servers while forwarding traffic inside the DC. Essentially, the results reveal a flexible and scalable routing mechanism where the number of signaling

messages and routing tables do not grow linearly with the size of the DC network, indicating the feasibility of using this proposal in large-scale DCs (ten of thousands of servers). Furthermore, it offers small stretch and adequate load distribution.

The remainder of this paper is organized as follows. Section 2 focuses on presenting the main server-centric related work which uses cube-based topologies. Section 3 details the proposed 3-cube server-centric DC networking architecture. Section 4 formalizes the proposed XOR-based flat routing mechanism. Section 5 presents the evaluations focusing on the performance levels achieved by the XOR-based flat routing mechanism in the proposed DC architecture. Section 6 brings the next steps and concludes the paper.

2. Related Work

The work presented in [Costa et al. 2009] proposes the concept of server-to-server communication, eliminating the use of intermediary switches and routers. The rationale is that since the DC infrastructure is owned and controlled by a single entity, it is easy to create topologies optimized to obtain the best performance while forwarding traffic through the network. In such work, routing is done on the Network Interface Card (NIC) itself, with packets forwarded directly from server NIC to server NIC by using a multi-port network interface card. In fact, there is a service responsible for processing and forwarding the packets between servers. This routing on the NIC approach is possible through commercially available NICs, capable of running computing capabilities [NIC 2010].

Another related work is presented in [Guo et al. 2009]. Such proposal also uses a server-centric approach, in which servers are equipped with multiple network ports that connect to few commodity and low cost mini-switches. The idea is to provide multiple parallel paths in order to increase the bandwidth and improve fault-tolerance.

Both works use the cube-based topology, which naturally offers a better path diversity in the network. By having such path diversity, the bisection bandwidth is increased, fault-tolerance is obtained, wiring complexity and the total cost are reduced, and load balance can be better exploited, potentially providing energy savings. The cube topology has been seen as an excellent alternative for attending the metrics mentioned above [Costa et al. 2009, Guo et al. 2009]. Although fat-trees topologies are commonly used in DC networks, they suffer from low resilience to failures and high wiring complexity [Costa et al. 2009]. Therefore, our work applies the XOR-based flat routing in 3-cube topologies focused on the server-centric approach.

3. Proposed 3-cube server-centric DC networking architecture

According to the networking architecture proposed in this paper, the DC is composed of a set of servers organized in a 3-cube topology, where all the servers are topologically distributed in three axis (x, y and z). In such server-centric scenario, servers are the fundamental elements in the DC, not requiring the use of other network elements, such as routers and/or switches to provide traffic forwarding during server-to-server communication. To this aim, each server comprises a general purpose multi-core processor, with memory, persistent storage and six NICs named from `eth0` to `eth5`. In order to be settled in the 3-cube topology, each server uses the NICs `eth0` and `eth1` in the x axis, the NICs `eth2` and `eth3` in the y axis, and the NICs `eth4` and `eth5` in the z axis, as illustrated in Figure 1.

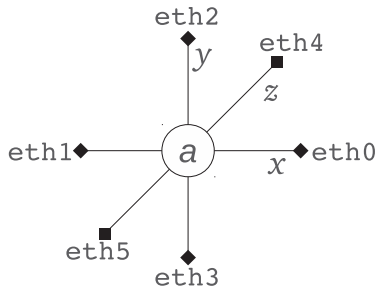


Figure 1. How to settle the NICs of server a in the 3-cube topology.

Considering the use of the NICs in each one of the three axis, wiring the proposed DC network become a simple task as exemplified in Figure 2. In this figure, there are three servers (a, b and c) located in the same row of the DC along the x axis. Basically, the required connections in a given axis are established using the two interfaces assigned to it, also considering servers located in the edges as neighbors. According to these procedures, it is possible to check the desirable organization in Figure 2, where `eth0` of server a is connected to `eth1` of server b, `eth0` of server b is connected to `eth1` of server c, and `eth0` of server c is connected to `eth1` of server a, wrapping the x axis to establish this last link. The same pattern is used for wiring the y and z axis, where `eth2` is connected to `eth3` and `eth4` is connected to `eth5`, respectively.

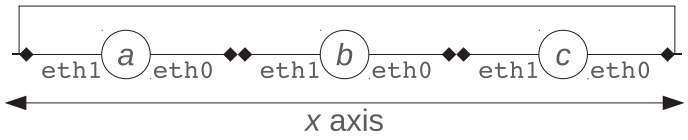


Figure 2. Wiring servers a, b and c in the x axis.

When compared to other topologies, the 3-cube topology not only offers simpler wiring and higher path redundancy, but also reduces to half the maximum distance (the radius) between servers, which is obtained by wrapping the links between the edge servers. In order to simplify the perception of the benefits of the links established between edge servers, Figure 3 presents a 3-cube topology composed of 27 servers. Note in this figure the existence of six NICs in all servers, the linkage between all the servers in the three axis, and the wrapping connections between the edge servers. Such wrapping connections raise the edge servers to a special condition in the DC, since they are able to invert traffic in the extremities of the axis.

Essentially, in the server-centric 3-cube topology, there are three different levels of edge servers as seen in Figure 3. The levels are relative to the ability that each edge server has to invert traffic along the three axis used in the topology. In the first level are the edge servers located in the inner parts of the 3-cube surface, labeled as S servers in the figure. These servers are able to perform traffic inversion in a single axis. After, in the second level are the edge servers located in the borders (B servers) of the 3-cube. These servers are able to perform traffic inversion in two axis. Finally, in the third level are the eight edge servers located in the corners (C servers) of the 3-cube. These eight servers are able to perform traffic inversion in all the three axis. In this way, when compared to other topologies, the 3-cube topology also offers higher resilience. For example, the

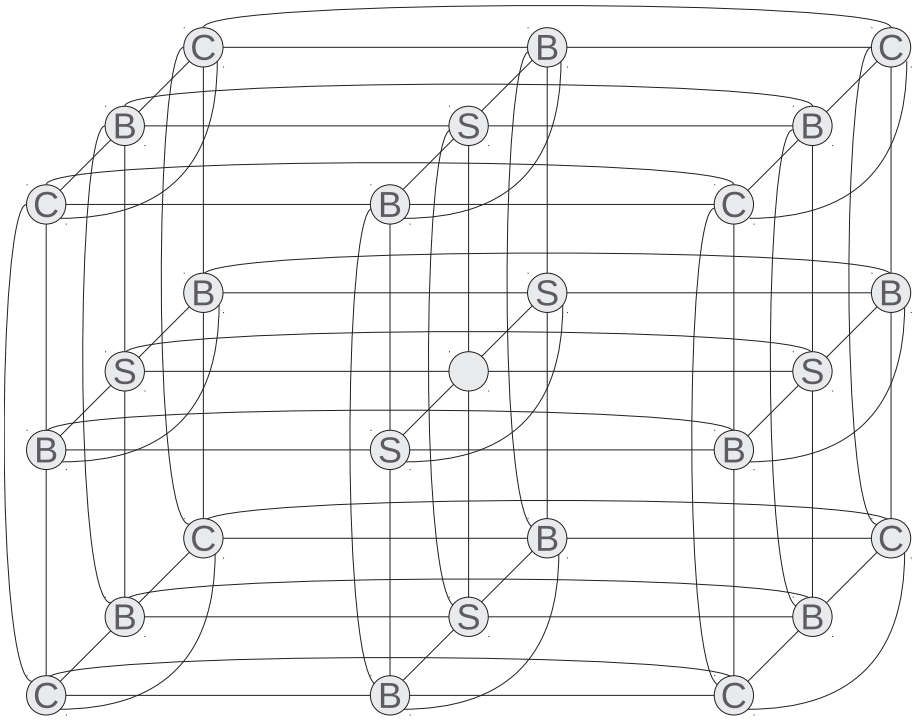


Figure 3. Example of a 3-cube topology where the axis are $x = 3$, $y = 3$ and $z = 3$.

DCell [Guo et al. 2004] network can be partitioned even if less than 5% of the servers or links fail. The server-centric 3-cube topology is able to operate even if nearly to 50% of the servers or links fail, its symmetrical and redundant structure allows failures in some regions of the topology, without impacting the performance of the remaining servers.

Regarding the DC configuration, there are available in the literature several techniques to bootstrap the configuration of all elements comprising the DC, such as servers, routers and switches. For example, in Portland [Mysore et al. 2009] it is used a hierarchical addressing scheme to assign *Pseudo MACs* to all servers according to their position in the DC. Afterwards, a *Location Discovery Protocol* (LDP) is used to help switches composing the fat tree topology to realize their role in the network, whether they are *edge*, *aggregation* or *core* switches.

In this work, due to the proposed XOR-based flat routing mechanism, uniqueness is the only requirement to assign flat identifiers to servers. Despite uniqueness, none topological adherence or semantic organization is required for the IDs. Actually, a total random distribution of IDs creates the ideal scenario for routing using the XOR-based mechanism. In this way, each server selects the smallest MAC address (among its six NICs) to be its 48-bits flat ID. Although such mechanism is simplistic, it suffices to achieve the routing mechanism requirements, and avoids the use of human intervention and/or bootstrap mechanism in order to configure the servers, i.e., all the required configuration is plug-and-play. Furthermore, switches and routers are not required to provide server-to-server communication, avoiding the use of special protocols to configure these equipments. However, if such equipments are used, they are intended to just offer a network bus connecting servers.

In order to perform server-to-server traffic forwarding, this proposal adopts MAC-

in-MAC encapsulation, due to the fact that the proposed flat routing mechanism is designed to operate using MACs as flat IDs of all servers inside the DC. Besides the legacy support to all applications running on upper layers of the network stack or inside virtual machines, the option for performing routing at the layer 2 in conjunction with the MAC-in-MAC technology has the advantage of fixing in two the number of packet headers from source to destination. Such characteristic simplifies the definition of the maximum payload size and avoids segmentation of packets on the path traversed inside the DC.

Figure 4 exemplifies the traffic forwarding from server a towards server e. In this figure, it is possible to verify that the inner header carries the server-to-server information (SRC Flat ID and DST Flat ID). This header is preserved during the entire packet transmission and, essentially, it is used by all the servers on the path to perform the XOR-based routing (detailed in Section 4). On the other hand, the outer header has only local link scope, being changed in all the links from source to destination. This header carries the MAC address of the server currently forwarding the packet, and the MAC address of the next server (next hop) present on the path towards the destination, i.e., it is used to transfer traffic between two adjacent servers.

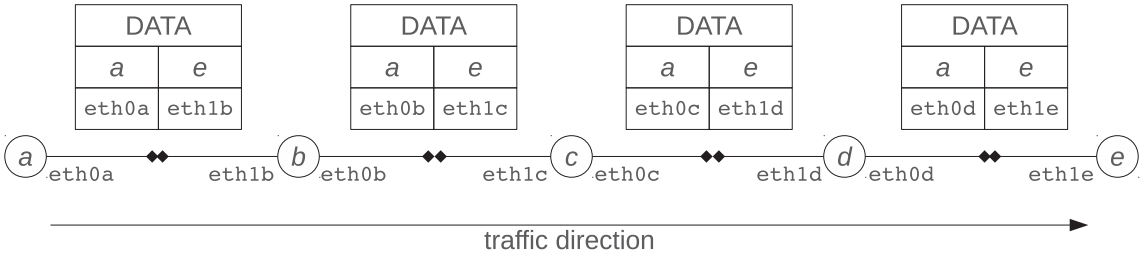


Figure 4. Traffic forwarding from server a to server e using MAC-in-MAC.

Finally, although it is kept out of the scope of this work, the proposed DC networking architecture predicts that among all the servers present in the DC, a fraction of the servers has one extra NIC to forward traffic in and out of the DC. For example, one possible solution for placing these nodes considers the edge servers, once they are able to invert traffic along the axis of the 3-cube topology. In this case, the designer of the DC can start by using servers located in the corners (C servers - 3 axis inversion), after servers located in the borders (B servers - 2 axis inversion) and, finally, servers located in the surface (S servers - 1 axis inversion) of the 3-cube topology.

4. XOR-based Flat Routing Mechanism

The proposed XOR-based routing discards the use of mapping services required in other proposals found in the literature [Greenberg et al. 2008, Greenberg et al. 2009, Mysore et al. 2009]. Usually, these mapping services provide source routing and/or tunneling information required to forward packets inside the DC network, characterizing an oblivious relation between servers and network structure.

The routing mechanism proposed uses n-bit flat identifiers to organize the routing tables in n columns and route packets through the network. Its routing principle uses the distance between two flat identifiers a and b as their bitwise exclusive or (XOR), which is represented by $d(a, b) = a \oplus b$, being $d(a, a) = 0$ and $d(a, b) > 0, \forall a, b$. Given a packet originated by server x and destined to server z, and denoting $\mathbb{Y}$ as the set of identifiers

contained on x's routing table, the XOR-based routing mechanism applied at server x selects the server $y \in \mathbb{Y}$ that minimizes the distance towards z, which is expressed by the following routing policy

$$\mathcal{R} = \underset{y \in \mathbb{Y}}{\operatorname{argmin}}\{d(y, z)\}. \tag{1}$$

Each server maintains a routing table in which its knowledge about neighbor servers is spread in n columns denominated buckets and represented by $\beta_i, 0 \leq i \leq n-1$. Table 1 presents an example of a routing table for server 0001 in an identity space where $n = 4$. Each time a server a knows a novel neighbor b, it stores that information in the bucket β_{n-1-i} given the highest i that satisfies the following condition[1]

$$d(a, b) \operatorname{div} 2^i = 1, a \neq b, 0 \leq i \leq n-1. \tag{2}$$

For example, consider $a = 0001$ and $b = 0010$. The distance $d(a, b) = 0011$ and the highest i that satisfies condition (2) is $i = 1$, concluding that the identifier $b = 0010$ must be stored in the bucket $\beta_{n-1-i} = \beta_2$. Basically, condition (2) denotes that server a stores server b in the bucket β_{n-1-i}, where $n - 1 - i$ is the length of the longest common prefix (*lcp*) existent between both identifiers of servers a and b. This can be observed in Table 1, where the buckets $\beta_0, \beta_1, \beta_2, \beta_3$ store the identifiers having *lcp* of length 0, 1, 2, 3 with server 0001. Such routing tables approach is one of the main advantages of the XOR-based mechanism, since a server only needs to know n (1 neighbor per bucket) of the possible 2^n servers available in the network to successfully route packets.

Table 1. Hypothetic routing table for server 0001 in an identity space where $n = 4$.

β_0	β_1	β_2	β_3
1000	**0**100	**00**10	**000**0
1001	**0**101	**00**11	
1010	**0**110		

The proposed mechanism considers a K factor which defines the minimum amount of information required per bucket. Since a bucket β_i has at most 2^{n-1-i} entries, if $K > 2^{n-1-i}$ we limit K for that bucket to $K = 2^{n-1-i}$. By varying K, servers have wider or narrower view of the DC network. In this way, during the process for building the routing tables, servers are aimed at finding K neighbors for each bucket. Essentially, a server a can know a server b from two distinct ways in the proposed process for building the routing tables: 1) a *discovery process* in which servers actively search for neighbors to fill the buckets and 2) a *learning process*, where servers use the information contained in the signaling messages that cross them to add more information to their buckets in a costless passive fashion.

4.1. Discovery Process

In the *discovery process* a `HELLO` message is used to insert servers directly (physically) connected in the routing tables, such servers are denominated as *physical neighbors*. Each

[1]div denotes the integer division operation on integers.

discovered *physical neighbor* is then stored in the bucket having the greatest i that solves condition (2). After introducing all the *physical neighbors* in the routing table, a server that still has buckets with less information than the defined K factor, actively searches for neighbors that can fill such buckets. Since such neighbors are not physically connected, they are called *virtual neighbors*.

In this way, servers start to send `QUERY` messages to the neighbors already stored in the buckets (in the initial step the buckets only contain *physical neighbors*), describing the buckets which still require information. The neighbor servers that receive the `QUERY` message and have at least one neighbor that fills in one of the requested buckets, answer with a `RESPONSE` message. After receiving the `RESPONSE` message, servers store the discovered *virtual neighbors* in the respective buckets. New queries are sent to the discovered neighbors if at least one of the buckets is still requiring information.

The following information are associated to each entry of the routing table: 1) the *physical neighbor* (next hop) towards the discovered neighbor, 2) the NIC used to establish the link with the *physical neighbor* (next hop) and 3) the physical distance in number of hops to the discovered neighbor. Afterwards, if a server a discovers a server b through different NICs, it can store all the information for path diversity purposes.

4.2. Learning Process

Figure 5 is used to explain the *learning process*. In this figure, server 1 sends a `QUERY` message to server 2, and server 2 sends back a `RESPONSE` message informing server 1 about the existence of server 3. Assuming that server 1 still needs information to satisfy the K factor in some of its buckets, it sends a `QUERY` message to server 3 just discovered, and server 3 sends back a `RESPONSE` informing server 1 about the existence of servers 4 and 5 (considering that both servers 4 and 5 satisfy what server 1 is looking for).

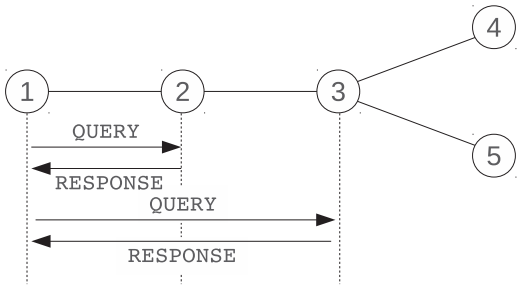


Figure 5. Exemplification scenario for the *Learning Process*.

The *learning process* takes place in two distinct moments. Firstly, it occurs when server 3 receives the `QUERY` message from server 1. At that instant, server 3 knows server 1 in a passive way, using the source ID present in the `QUERY` message. Secondly, the *learning process* occurs when server 3 sends the `RESPONSE` message to server 1 containing information about servers 4 and 5. In this case, the `RESPONSE` message crosses server 2 which, also passively, learns about both servers based on the information contained in the `RESPONSE` message destined to server 1. Essentially, the *discovery* and *learning processes* were designed to assure symmetry between the routing tables of servers in the DC network, as defined in Property 1. Afterwards, the process for building the routing tables is designed to prioritize the discovery of neighbors physically near (in

number of hops), contributing in this way to the convergence of the routing tables, and reducing the required number of signaling messages.

Property 1 *If a server **a** has a server **b** in its routing table, then server **b** has server **a** in its own routing table.*

4.3. Routing Process

Each time a server receives/generates a packet to forward, it tries to route the packet by applying the routing policy $\mathcal{R}$ defined in (1). The routing process starts by identifying the bucket to be used. Considering that a server a is routing a packet destined to b, as exemplified in Figure 6, it defines the bucket β_i, where the index i is the highest one that solves condition (2). Then server a routes the packet to the server available in its routing table that is closest to server b. In other words, the packet is routed to n_R, which is selected by server a computing the following solution

$$n_R = \underset{id \in \beta_i}{\operatorname{argmin}}\{d(id, b)\}, \tag{3}$$

where id represents each one of the identifiers contained in the bucket β_i.

Since n_R is the identifier of a server that can be a *virtual neighbor* $j-$hops away ($j > 1$), server a uses the information associated to n_R in its routing table entry to forward the packet. Essentially, the information associated is the physical next hop g and the NIC linking both servers (previously detailed in Section 4.1), which are resultant of the *discovery* and *learning* processes for building the routing tables.

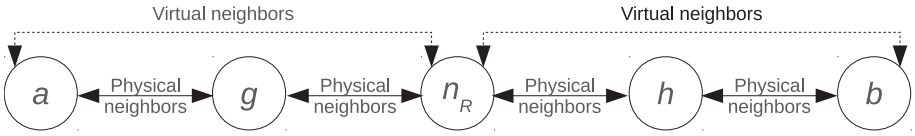


Figure 6. Example of the XOR-based routing process.

When the packet arrives at g, it also computes (3) and, again, due to the process for building the routing tables, it finds n_R in its bucket β_i, delivering the packet to n_R. Finally, n_R is the server which represents progress towards the destination server b in the flat identity space, i.e., the server pointed by the previous routing decision, firstly taken at source server a, which reduces the distance towards server b according to the routing police $\mathcal{R}$ defined in (1). In this way, based on the example of Figure 6, server n_R computes (3) and finds the destination server b in its routing table, towards its NIC connected to the physical neighbor h. Consequently, the remaining actions are responsible for delivering the packet to server b passing through server h.

5. Evaluations

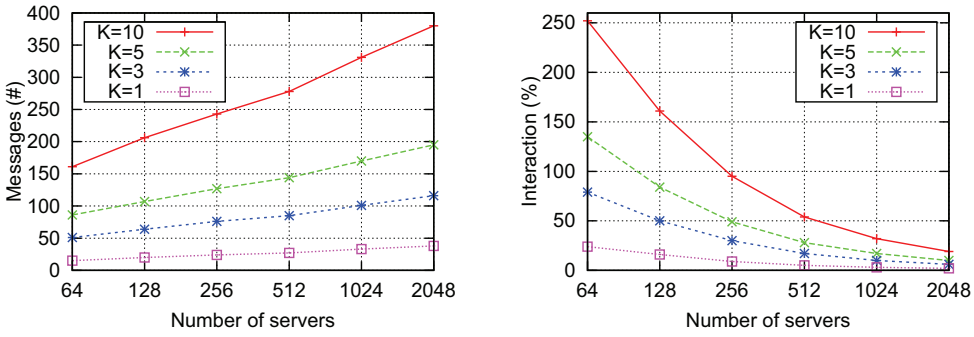
This section presents the evaluations of the proposed XOR-based flat routing mechanism in the proposed 3-cube server-centric DC networking architecture. The main objective of this section is to demonstrate, through evaluations using a developed emulation tool, the level of scalability achieved by the XOR-based routing mechanism in terms of number of entries in the routing tables. Such information is essential to demonstrate the feasibility

of integrating servers and networking, creating a server-aware DC architecture for the deployment of services, opposing the oblivious approach found in the literature.

This section also details the required signaling to converge the flat routing mechanism, presents the route stretch incurred in such scenario of reduced routing tables, and depicts the load distribution among all the servers during traffic forwarding inside the proposed 3-cube DC network. These evaluations were performed using six networks of different sizes – 64, 128, 256, 512, 1024 and 2048 servers – using the K factor set to 1, 3, 5 and 10, resulting in more than 22 million computed paths, i.e., it was computed the full combination of source/destination paths.

The developed emulation tool has an important contribution in the evaluations, since it offers the possibility of emulating individual servers provided with a full implementation of the XOR-based flat routing mechanism. Basically, each server inside the 3-cube DC is instantiated as an individual thread, being able to perform all the required interactions between the servers, exchanging signaling messages to build the routing tables according to the protocol specification. Afterwards, the threads (the emulated servers) are able to forward traffic inside the 3-cube DC, contributing with the analyzes related to route stretch and load distribution. The tool also includes a topology generator capable of creating 3-cube topologies, considering all the required links and assuring the occurrence of uniqueness and randomness while assigning the flat IDs.

The first results are presented in Figure 7, where it is possible to find information regarding the signaling complexity required to converge the XOR-based routing tables. Figure 7(a) describes the average number of signaling messages generated per server in all the six evaluated topologies. This number represents the exchange of `QUERY` and `RESPONSE` messages from server-to-server during the *discovery process*, and shows that as the size of the topologies and the value of the K factor increase, the number of signaling messages also increases.

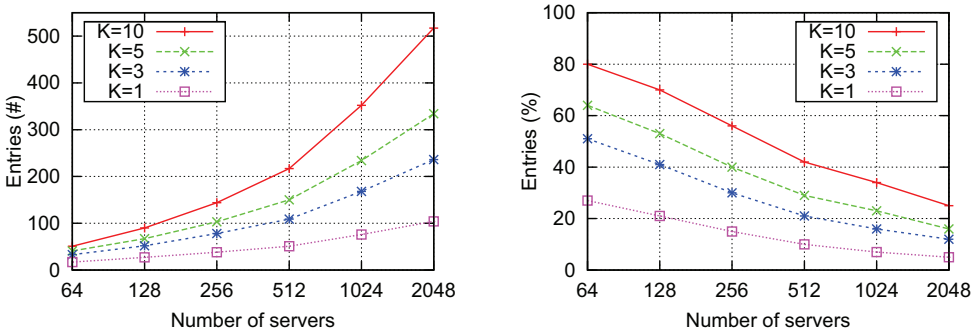


(a) Number of messages exchanged. (b) Percentage of interaction.

Figure 7. Results regarding the average signaling exchanged per server.

However, analyzing the percentage of interaction between the servers present in the six topologies, it is possible to verify that the percentage of interaction between the servers does not increase linearly with the size of the topology, as shown in Figure 7(b). This behavior has origin in the proposed *discovery* and *learning processes* designed to extract the maximum information from the signaling, prioritizing the exchange of messages with servers located physically near to the requesting server.

The next result detailed in Figure 8 presents the average number of entries required in the routing tables of all servers. Similar to the results obtained for the signaling messages, the bigger the network and the K factor used, the higher the number of entries present in the routing tables, as seen in Figure 8(a). However, the percentage of routing information required in the routing tables also decreases as the network topologies increase, as presented in Figure 8(b). The main reason for this behavior is the proposed XOR-based routing tables which require only a small fraction of the entire routing information available in the network, in order to assure traffic delivery.



(a) Number of entries in the routing tables. (b) Percentage of network knowledge.

Figure 8. Results regarding the average routing tables per server.

The results presented in Figure 8 not only demonstrate that the proposed flat routing mechanism is able to operate with a fraction of the overall routing information, offering better control for the rate at which the routing tables grow, but also prove its ability for integrating servers and network structure in a scalable manner, offering the required infrastructure for inserting information regarding the servers composing the DC in the routing tables. Such condition is ideal for the development of new services totally integrated with the available DC network structure.

Normally, based on the small number of entries present in the routing tables, it is expected that the proposed XOR-based flat routing mechanism does not find the shortest paths for all the server-to-server communication cases. Basically, route stretch is a trade-off of the number of entries present in the routing tables. Nonetheless, Figure 9 details the obtained stretch values, achieving values near to optimal (stretch 1). For example, considering the results obtained for the topology with 2048 servers using $K = 1$, the servers exchanged approximately 40 signaling messages (see Figure 7(a)), corresponding to the interaction with 2% of the servers, in order to create routing tables with approximately 100 entries (see Figure 8(a)). Such number of entries represents a knowledge of 5% of the entire network, and even with this reduced number of entries the protocol is able to deliver traffic with an average route stretch around 1.85, not even doubling the length of the shortest paths available. In the sequence, according to the basic trade-off mentioned, in the results obtained for the topology with 2048 servers using $K = 10$, it is possible to check the bigger routing tables, containing approximately 25% of the entire routing information, and the reduced stretch at the level of 1.17.

Finally, the next results demonstrate the load distribution among all the servers. Figures 10(a), 10(b), 10(c) and 10(d) detail the load for all the evaluated topologies using

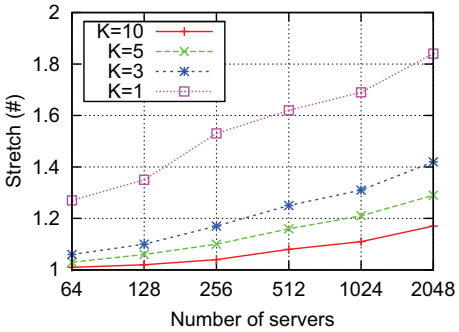


Figure 9. Results for route stretch.

the K factor set to 1, 3, 5 and 10, respectively. Essentially, the results indicate that approximately 80% of all servers, independent of the number of servers in the DC and the K factor used, operate at a load level ranging from 0.8 to 1.2, i.e., most of the servers deviate only 20% from the average traffic forwarding load. Firstly, such numbers indicate the robustness of the 3-cube topology, a topology capable of providing an elevated path redundancy ideal not only for load distribution, but also for fault resilience. Afterwards, such numbers prove the perfect coupling between the 3-cube topology and the proposed XOR-based flat routing mechanism, indicating that the plug-and-play scenario of totally random distributed flat IDs is ideal to spread the forwarding load among all the servers, not even requiring the usage of other load balancing mechanism, such as VLB or ECMP.

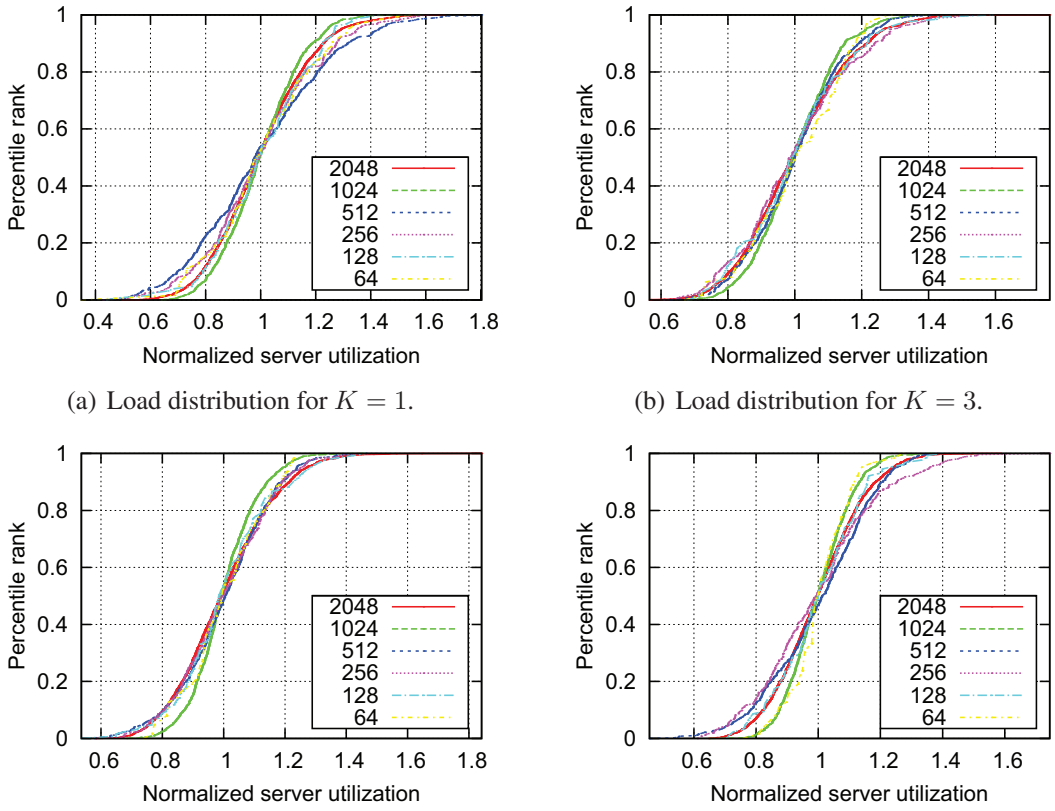
6. Conclusion and Future Work

This paper presented a new server-centric DC architecture where servers are organized in a 3-cube topology. In this work, servers are the main elements of the DC, eliminating the need for traditional network equipments such as switches and routers in order to provide server-to-server communication. The main novelty is the integration between servers and networking, which is done by the scalable XOR-based flat routing mechanism.

Among the benefits of the 3-cube topology are the higher path redundancy and the simpler wiring. The establishment of wrapped links between the edge servers significantly reduce the maximum distance between two servers, and increases the resilience of the DC. Essentially, the edge servers are able to invert traffic in the three axis, assuring that the DC operates even if approximately 50% of servers and/or links fail. The 3-cube topology in conjunction with the XOR-based flat routing mechanism creates a plug-and-play scenario, efficient for load distribution among all the servers in the DC, independent of the size of the topology and of the K factor used.

Once the proposed routing mechanism requires only a fraction of the entire routing information, it provides the total integration between servers and network structure, eliminating the concept of a black box network. As presented in the evaluations in this paper, as the DC network grows in number of servers, the required signaling and the number of entries in the routing tables do not linearly follow such growth. At the same time, although the routing tables have just a small percentage of information, the paths from source to destination present a small stretch penalization.

As future work, one objective is to investigate the connectivity between 3-cubes.



(a) Load distribution for $K = 1$.

(b) Load distribution for $K = 3$.

(c) Load distribution for $K = 5$.

(d) Load distribution for $K = 10$.

Figure 10. Results for load distribution among all servers.

Nowadays, it is expected that enormous DCs (tens of thousands of servers) are modularized, simplifying their deployment and maintenance. For example, it is possible to ship these DCs in containers, simplifying their transport and reducing cots regarding energy and cooling systems, as proposed in [Guo et al. 2009, Wu et al. 2009]. The idea is to use some specific servers for inter-3-cube communication through fiber channels.

Acknowledgment

The authors would like to thank Ericsson Brazil and CNPq for supporting this work.

References

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2006). BigTable: A Distributed Storage System for Structured Data. *In Proceedings of OSDI 2006, Seattle, WA, USA*.

Cisco (2010). Cisco Unified Computing System. Available at `http://www.cisco.com/go/unifiedcomputing`.

Costa, P., Zahn, T., Rowstron, A., O'Shea, G., and Schubert, S. (2009). Why Should we Integrate Services, Servers, and Networking in a Data Center? In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN'09)*, pages 111–118, New York, NY, USA. ACM.

Dean, J. and Ghemawat, S. (2008). DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. *In Proceedings of ACM SIGCOMM 2008, Seattle, WA, USA*.

Esteve, C., Pasquini, R., Verdi, F. L., and Magalhães, M. F. (2010). Novas Arquiteturas de Data Center para Cloud Computing. *Book Chapter published as a Mini Course of the 28th Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2010), Organized by: C. A. Kamienski; L. P. Gaspary; M. P. Barcellos. ISBN: 9772177497006. Idiom: Portuguese. Gramado, RS, Brazil.*

Ghemawat, S., Gobioff, H., and Leung, S. T. (2003). The Google File System. *In Proceedings of SOSP 2003, Bolton Landing (Lake George), New York, USA*.

Google (2010). Google's Custem Web Server, Revealed. Available at `http://tinyurl.com/google-custom-server`.

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2009). VL2: A Scalable and Flexible Data Center Network. *In Proceedings of the ACM SIGCOMM 2009 - Conference on Data Communication, Barcelona, Spain*.

Greenberg, A., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2008). Towards a Next Generation Data Center Architecture: Scalability and Commoditization. *In Proceedings of the ACM Workshop on Programmable Routers For Extensible Services of Tomorrow, Seattle, WA, USA*.

Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. (2009). BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. *In Proceedings of the ACM SIGCOMM 2009 - Conference on Data Communication, Barcelona, Spain*.

Guo, C., Wu, H., Tan, K., Shiy, L., Zhang, Y., and Luz, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *In Proc. of OSDI 2004, San Francisco, CA, USA*.

Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. (2009). PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. *In Proceedings of the ACM SIGCOMM 2009 - Conference on Data Communication, Barcelona, Spain*.

NIC, K. (2010). Killer NIC. Available at `http://www.killernic.com`.

Pasquini, R., Verdi, F. L., and Magalhães, M. F. (2009). Towards a Landmark-based Flat Routing. *27th Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2009), Recife, PE, Brazil*.

Pasquini, R., Verdi, F. L., Magalhães, M. F., and Welin, A. (2010a). Bloom filters in a Landmark-based Flat Routing. *In Proc. of IEEE ICC 2010, Cape Town, South Africa*.

Pasquini, R., Verdi, F. L., Oliveira, R., Magalhães, M. F., and Welin, A. (2010b). A Proposal for an XOR-based Flat Routing Mechanism in Internet-like Topologies. *In Proceedings of IEEE GLOBECOM 2010, Miami, FL, USA*.

Wu, H., Lu, G., Li, D., Guo, C., and Zhang, Y. (2009). MDCube: A High Performance Network Structure for Modular Data Center Interconnection. *In Proceedings of the ACM CONEXT 2009, Rome, Italy*.