Free-Scaling Your Data Center

László Gyarmati, András Gulyás, Balázs Sonkoly, Tuan A. Trinh
Budapest University of Technology and Economics
Department of Telecommunications and Media Informatics
Email: {gyarmati,gulyas,sonkoly,trinh}@tmit.bme.hu

Gergely Biczók
Norwegian University of Science and Technology
Department of Telematics
Email: gbiczok@item.ntnu.no

***Abstract*—The increasing popularity of both small and large private clouds and expanding public clouds poses new requirements to data center (DC) architectures. First, DC architectures should be incrementally scalable, i.e., the architecture should support the creation of DCs of arbitrary size while retaining key performance characteristics. Second, initial DC deployments should be incrementally expandable, i.e., the architecture should support small-scale upgrades without a notable decrease in operation efficiency. A DC architecture possessing both properties satisfies the requirement of *free-scaling*.**

Recent work in DC design focuses on traditional performance and scalability characteristics, therefore resulting in symmetric topologies whose upgradability is coarse-grained at best. In our earlier work we proposed Scafida, an asymmetric, scale-free network inspired DC topology which scales incrementally and has favorable structural characteristics. In this paper, we build on Scafida and propose a full-fledged DC architecture achieving free-scaling. Our main contribution is threefold. First, we propose an organic expansion algorithm for Scafida; this combined with Scafida's flexible original design results in a freely scalable architecture. Second, we introduce the Effective Source Routing mechanism that provides near-shortest paths, multi-path and multicast capability and low signaling overhead by exploiting the benefits of the Scafida topology. Third, we show based on extensive simulations and a prototype implementation that Scafida is capable of handling the traffic patterns characteristic of both enterprise and cloud data centers, tolerates network equipment failures to a high degree, and allows for high bisection bandwidth.

***Index Terms*—data center, scale-free, routing, fault-tolerance**

I. Introduction

Fueled by the paradigm change to cloud computing, data centers (DCs) are all the talk both in networking practice and research today. So far, large public cloud data centers have been dominating the scene with the likes of Google, Amazon, Microsoft, etc. building out and maintaining data centers consisting of tens or hundreds of thousands of servers, serving the need of billions of users. However, another trend is gaining momentum: more and more organizations decide to consolidate their computing resources into small- or medium-scale private clouds [11]. There are multiple reasons behind the surge of private clouds. First, security and privacy issues using a public infrastructure can be prohibitive for certain organizations, such as governments [10]. Second, private cloud operation policies and management procedures can be tailor-made to the owner's liking [11]. Finally, of course, cost is always a deciding factor; surprisingly, operating a private cloud could be advantageous in the long(er) run [1]. As a consequence, the increasing proliferation of both large and small data centers are highly likely.

Both large and small cloud structures would greatly benefit from *free-scaling*, i.e., to be able to build and operate a data center of arbitrary size, and constantly upgrade it during its lifetime. It is straightforward to see the benefit for small companies: they would be able to start a private data center quickly and with a limited budget, and build it out incrementally [22]. Note that even if servers are co-located at a data center provider, SMEs still have to face increasing costs as the number of servers increases. On the other hand, upgrades are also frequently needed in large data centers, triggered by a growing user base (private, e.g., Facebook) or deployment of more demanding cloud applications and getting more corporate customers (public, e.g., Amazon). As a well-known real-world example, Facebook has upgraded its data center facilities frequently and step-by-step [24], resulting in a doubling of servers over the course of 7 months in 2010 [25]. Free-scaling demands two separate properties from a data center architecture. First, the architecture should be *incrementally scalable*, i.e., it should be possible to build a data center with $1,000$ or $1,100$ nodes, while retaining the same performance indicators. Second, the data center should be *incrementally expandable*, i.e., frequent small-scale upgrades (adding 10 servers or larger switches) should be feasible and result in a comparably efficient data center. If coupled with the ability to use commodity switches for connecting servers (already partly available on site), such a free-scaling architecture would allow system administrators to build actual DC networks in a do-it-yourself manner. Moreover, if the processing and storage requirements of the given organization change over time, it would be possible to reuse existing equipment when expanding the network.

Albeit networking infrastructure issues inside a DC have received tremendous attention from the research community in recent years, as evidenced by innovative designs both for network fabric [12], [26] and topology [2], [14], [13], with scalability, fault-tolerance, and high bisection bandwidth in focus, far less emphasis has been put on incremental properties. For example in case of recursive DCell [14], it is required that "the minimal quantum of machines added at one time be much larger than one". Generally speaking, state-of-the-art DC topologies need to preserve their *symmetry* in order to achieve their demonstrated high performance. Symmetry in turn, especially for sophisticated topology concepts, results in the type of scalability which only manifests itself at well-defined network sizes. MDCube [34] was designed to provide a way to create DCs of different size, its coarse-grained scale (around 1,000 servers per box), differing requirements (being economical is more important then being power-dense) and the foreseeable price of hardware repair and software upgrade (likely carried out by transporting whole containers back to their respective vendors) can be prohibitive for SMEs on a budget [20]. LEGUP [8] leverages hardware heterogeneity to reduce the cost of upgrades in fat-tree based data centers. Jellyfish [32] proposes a random graph topology to facilitate upgradability of data centers; an interesting idea, however, Jellyfish sacrifices structure creating a significant challenge for their future routing mechanism.

In our earlier work we proposed *Scafida*, an asymmetric, scale-free network (SFN) inspired data center topology [16]. At the structural level, Scafida exploits the inherent small diameter and fault-tolerance of scale-free networks, while it copes with the number and physical constraints (i.e., available ports) of available servers and switches. The node degrees are limited by extending the original preferential attachment algorithm of Barabási and Albert [5]. The Scafida structure (while preserving the favorable properties of scale-free networks) can be adjusted on a fine-grained scale regarding network size, and performs comparably well to the recently proposed symmetric DC topologies at various network sizes. Scafida satisfies the incremental scalability requirement; however, in [16] we focused on the justification of the topology itself, leaving practical data center operational aspects such as incremental expansion, routing and implementation for future work.

In this paper we address these very issues and extend the Scafida topology to a full-blown data center architecture. Our main contribution is threefold:

1) We propose an **organic expansion algorithm** with which it is possible to upgrade an existing Scafida data center in fine-grain increments. The expanded data center performs almost identically to the original one. Put together with its incremental scalability, Scafida satisfies the requirements of **free-scaling**.
2) At the networking level, we propose **Efficient Source Routing (ESR)**, inspired by efficient paths in complex networks, which provides near-shortest paths, multi-path capability, efficient multicasting and low signaling overhead. The proposed routing mechanism together with the underlying topology demonstrates the ability to efficiently handle the traffic pattern characteristic of both cloud and enterprise data centers, to tolerate network equipment failures to a high degree, and to allow for high bisection bandwidth.
3) We design and implement a **prototype** for Scafida. We validate the operation of the prototype both in a virtual test network with several hundreds of nodes and a live testbed with firmware-modified commodity switches.

The rest of the paper is organized as follows. Section II gives an overview of the Scafida topology and its favorable structural properties for flexible data center usage. Section III proposes an incremental expansion mechanism. Section IV describes ESR including addressing and forwarding with Bloom filters, the basic routing algorithm and fault-tolerance. Simulation results are shown demonstrating the routing performance of our design. Section V describes our prototype implementation and provides validation experiments. We discuss important cost implications of our design in Section VI, while related work is briefly presented in Section VII. Finally, we conclude in Section VIII.

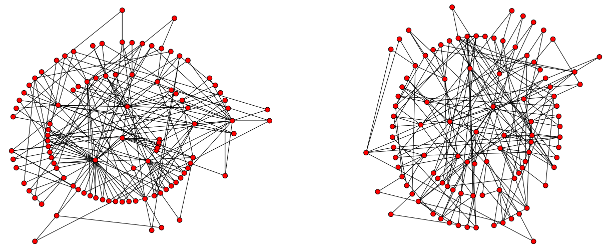

(a) Original scale-free network (b) Scafida with 8 as the maximum degree

Fig. 1. Scale-free vs. degree-limited Scafida networks [16]

II. Basics of the Scafida Structure

This section reviews the basic properties of the Scafida data center structure. First, we briefly review our topology generation algorithm and the properties of the generated topology. Afterwards, we discuss Scafida's scalability in both the traditional and the incremental sense.

A. Basic properties

The goal of this part is to summarize the basic properties of Scafida data center topologies including structural performance characteristics; for additional details please refer to [16]. By structural, we mean properties which are independent of routing, virtualization techniques, and actual application types.

Structure generation. Our topology generation algorithm is a modified version of the preferential attachment algorithm of Barabási and Albert [5]. The network structure is generated iteratively, i.e., nodes are added one by one; a new node is attached to an existing node with a probability proportional to the existing node's degree. Our method artificially constrains the number of links that a node can have, i.e., the maximal degree of the nodes, in order to meet the port number of servers and switches as commodity switches usually come with 4, 8, 16, 24, or 48 ports. We illustrate the difference between scale-free and Scafida networks in Figure 1.

Path length. As the servers of a data center communicate with each other, the average length of the paths between the nodes can fundamentally impact the performance of the data center network. In [16] we demonstrated that irrespective of the size of the networks, the average path lengths increase moderately with stricter constraints on the maximum degree.

Bisection bandwidth. Some applications (e.g., MapReduce [9]) require intensive communication among the servers of the data center; bottlenecks in the topology would cause performance degradation. The throughput capability of a data center can be measured with bisection bandwidth. In case of Scafida, the degree limitation does not effect the bisection bandwidth values of the topologies.

Fault tolerance. Due to their specific structure scale-free networks tolerate the random failure of nodes well [3]. This resilience comes handy in data centers, where random failures are bound to happen due to the large number of nodes. On top of that, constrained Scafida demonstrates even better behavior since there is no high-degree switch that can fail; therefore, a possible failure affects only a smaller part of the network. As Scafida data centers can be built using commodity switches similar to state-of-the-art DCs, Scafida is not more vulnerable to network failures—including attacks—than state-of-the-art systems.

Comparison to the state-of-the-art. Recently proposed data center architectures trade off incremental scalability and expandability for performance. Scafida can achieve comparable structural performance, while still being freely scalable. Due to its flexibility, Scafida networks can be created out of the parts of given state-of-the-art data centers. Accordingly, the performance of the topologies can be compared. Scafida has similar bisection bandwidth and average shortest path lengths as state-of-the-art architectures.

B. Scalability

Traditional scalability. Scafida scales extremely well in the traditional sense. In order to quantify the impact of network size on important benchmarks, we carried out extensive simulations on data center topologies of different sizes and degree constraints. Mean values on average shortest path length (SPL), network diameter (D) and bisection bandwidth (BB) are shown in Table I (50 simulation runs). Note that standard deviation values of SPL and D are around 1 and independent of network size, while standard deviation values of BB are around 21 (1000 nodes), 50 (5000 nodes), and 325 (10000 nodes), respectively, and independent of degree constraints. Both average shortest path length and diameter scales logarithmically, while bisection bandwidth scales linearly with respect to network size. On the other hand, constraining maximum node degrees has only a moderate effect on performance; the comparison of the two extremes (8 vs. No Limitation) shows only minor differences.

Incremental scalability. The first requirement of free-scaling is that a proper topology should be generated out of any reasonable number and types of servers and switches. Due to its *iterative, node-by-node* topology construction, and its capability to take hardware constraints into account, Scafida satisfies this requirement at the structural level. Note, that we assume a reasonable ratio of switches and servers and a satisfactory number of ports (requirements for any data center topology). In the next section, we demonstrate that Scafida can also be expanded incrementally without sacrificing the beneficial properties of the topology.

III. Incremental expansion

The structure of a freely scalable data center should also allow for *organic expandability*: any reasonable number of switches and servers could be added to an already existing network while retaining its preferable characteristics. Next, we propose an *iterative* algorithm that can increase the size of Scafida data centers efficiently. The method extends a Scafida topology by adding any given set of new switches and servers to the network. Additional switches are not added to the edge of the topology, instead they are replacing existing devices with fewer ports. The pseudocode of the expansion algorithm is presented in Figure 2. The algorithm works on the logical topology, represented by the set of nodes ordered by their degrees (from servers through small switches to large switches). In a bottom-up manner, it determines which new switches should replace devices lower in the degree-hierarchy, demoting the replaced switches to the next tier. Sequential expansion points are determined probabilistically giving much larger chance to higher degree nodes (this facilitates placing larger switches in the middle of the topology). Contrary to a top-down approach, the bottom-up construction assures that even servers could be replaced with a large switch at the end of a replacement series; however, the probability of this event is low.

As an example, suppose we would like to expand a topology consisting of switches with 4 and 8 ports, with a 16-port switch. The switch replacement process happens in a bottom-up approach. Namely, first we determine randomly which server will be replaced with a 4-port switch, then a random 4-port switch is replaced with an 8-port switch, finally a randomly selected 8-port switch is replaced with the 16-port switch. After the new switches are added to the topology, additional servers can be deployed by connecting them to the topology based on the preferential attachment principle.

To quantify the goodness of the method, we incrementally expanded the size of 1000-server Scafida data centers then we compared the properties of the expanded data centers with Scafida networks, freshly generated out of the same set of network elements. The results in Figure 3 indicate that Scafida data centers can be organically and efficiently expanded as the performance properties of the networks are nearly identical.

Put together, demonstrated incremental scalability and organic expandability equal to the Scafida structure eventuates free-scaling.

IV. Efficient Source Routing

Here we propose the Efficient Source Routing (ESR) mechanism that exploits the favorable properties of the Scafida topology, while addressing the routing challenge posed by the asymmetric and heterogeneous structure. ESR applies the paradigm of source routing motivated by the following reasons. First, source routing offers tight performance management and multipath capability for load-balancing. Second, a data center is typically operated by a single organization; therefore, there are no prohibitive, Internet-related security problems, and global topology is known by the operator. Third, the topology of a data center is static, especially compared to the Internet.

In this section, first, we introduce the applied addressing scheme, second, we describe the path selection procedure and error recovery, and finally, we evaluate routing performance by simulation.

TABLE I
SCALING PROPERTIES OF SCAFIDA WITH DIFFERENT DEGREE CONSTRAINTS (SPL MEAN, D MEAN, BB MEAN)

Size	8	16	24	48	No Limitation
1000	5.03, 9.5, 1000.38	4.58, 8.9, 999.59	4.41, 8.7, 999.97	4.18, 8.0, 1001.31	4.06, 7.9, 1001.79
5000	6.15, 11.1, 5007.74	5.54, 10.0, 5009.48	5.30, 9.9, 5008.01	5.02, 9.8, 5008.32	4.73, 8.9, 5010.69
10000	6.62, 12.0, 10339.72	5.93, 11.0, 10298.38	5.67, 10.2, 10314.56	5.37, 9.9, 10313.38	5.01, 9.9, 10299.53

```
Input:
$G = (V, E)$ — the Scafida topology
$n_{t_0}$ — number of new servers
$p_{t_0}$ — number of servers' ports
$n_{t_1}, \dots, n_{t_k}$ — number of new $t_i$ type switches
$p_{t_1}, \dots, p_{t_k}$ — number of ports of $t_i$ type switches
$a_{t_i}$ — number of $t_i$ type switches already added
$d_v$ — degree of the node $v \in V$
$b_v$ — maximal possible degree of the node $v \in V$
$m$ — number of links a newly added node has

Algorithm
1  $a_{t_i} = 0 \; \forall i$
2  for $v \in V$ do
       // determine the type of the nodes
3      $b_v = p_{t_i}$ where $p_{t_{i-1}} \leq d_v \leq p_{t_i}$
4  for $i \in [1, \dots, k]$ do
5      while $\sum_{j \geq i} a_{t_j} < \sum_{j \geq i} n_{t_j}$ do
6          $R = \{\}$ // feasible expandable nodes
7          for $v \in V$ do
8              if $b_v = p_{t_{i-1}}$ then
9                  $R = R \cup \{v\}$
10         if $i = 1$ then
11             $n_{t_0} + = 1$ // an additional node to be added
12         $c = random(R)$ // a random node
13         $b_c = p_{t_i}$ // expand the node
14         $a_{t_i} + = 1$
   // add the new nodes
15 for $i = 1, \dots, n_{t_0}$ do
16     $R = \{\}$ // for preferential attachment
17     for $v \in V$ do
18         if $b_v > d_v$ then
19             for $j = 1, \dots, b_v - d_v$ do
20                 $R = R \cup \{v\}$
21     while $|T| < m$ do
22         repeat
23             $v_t = random(R)$ // a random item of R
24         until $v_t \notin T$
25         $T = T \cup \{v_t\}$
26     $V = V \cup \{w_i\}$ // add a new node
27     for $v \in T$ do
28         $E = E \cup \{(w_i, v)\}$ // add a new edge
```

Fig. 2. The expansion algorithm

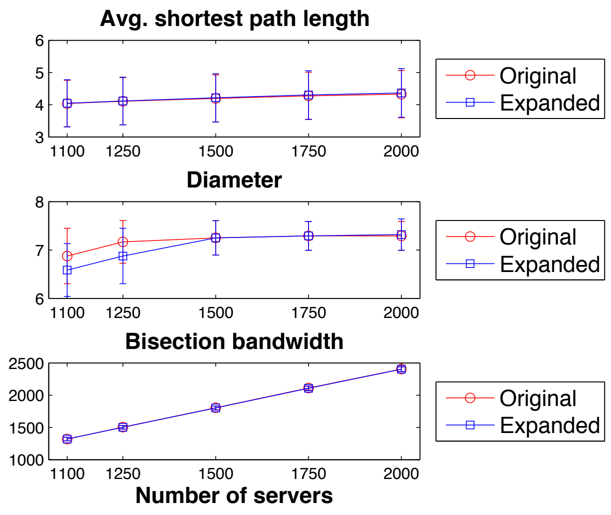


Fig. 3. Scafida can be expanded efficiently

A. Addressing and Forwarding: Bloom Filters

Since they have stochastically heterogeneous topologies, routing on traditional IP addresses would be inefficient in Scafida networks. As node addresses cannot be aggregated properly, routing tables sizes would be large, which contradicts the goal of applying low-end, commodity network equipments.

Accordingly, ESR addressing is based on *Bloom filters* [7]. Unique identifiers are generated for every link in the topology; these are the Bloom IDs of the links. In order to create a Bloom address for a target server, every single link identifier along the chosen path is summed up using the bitwise OR operator, from source to target. The resulting Bloom address, stored in the packet header, contains the routing information for the packet, i.e., the used links are implicitly stored. Bloom addresses have local validity, meaning that a demand can be routed to the specific target only from the source server where the Bloom address was generated. Note, that address size is fixed for all path lengths.

As the packet traverses through the network, along with its routing information coded into the Bloom address, intermediate switches inspect it. Any given switch compares the address of the packet with the IDs of their outgoing interfaces; the packet will be forwarded on the matching links (except for the incoming link, to avoid loops). Note, that the forwarding procedure inherently supports multicast; the source node just has to code all desired link IDs into the Bloom address. Network switches execute the bitwise AND operation exclusively, which has modest resource requirements. This is in line with the implications of recent data center traffic measurements [6], i.e., due to the high arrival rate of the flows, forwarding/routing decisions have to be made quickly.

False positive matches can occur in case of applying Bloom filters. Should such a false positive arise at a switch, the packet would also be forwarded on an additional link, creating unnecessary overhead and may also eventuate forwarding loops. The probability of such an occurrence depends on the size of the ID space; i.e., if the links are labelled by longer IDs, the possibility of a false positive match decreases. As an illustration, we computed the applicable Bloom ID lengths for different data center sizes assuming that the probability of false positive matches cannot exceed 10^{-4}. Thus, the lengths of Bloom IDs are 59, 73, 87, and 102 in case of DC with 500, 1000, 5000, and 10000 servers, respectively. A relatively small Bloom filter can eventuate a small false positive probability. In addition, Scafida can benefit from false positive-free operation by choosing false positive free paths exclusively similarly as in [30], employ smart techniques to avoid loops [31] or utilize soft-states in switches.

Clearly, Bloom filters can efficiently accommodate the variable length of paths in Scafida data centers. In addition, the application of Bloom filters offers built-in multicast support. Multicast emerges as a crucial routing level capability for data centers, since numerous cloud applications include routines where the same data have to be shared with multiple other servers (1-to-n pattern); MapReduce [9] being the best known example. In Scafida, creating multicast packets is as easy as creating unicast packets; the Bloom IDs of all desired links should be incorporated into a single Bloom address.

B. Path Computation

The Scafida structure is inspired by scale-free networks; thus, the path computation mechanism of ESR is based on the *efficient routing* algorithm [35] that has been designed for networks generated with the Barabási–Albert method. It avoids overloading high-degree network nodes (so-called hubs) by weighting links with the degree of the nodes; the weighting factor is denoted by β. Accordingly, traffic load is spread in the network resulting in improved available bandwidth conditions, and hence decreasing oversubscription—a key metric in today's data center networks.

In order to exploit and adapt to the properties of the Scafida structure, we slightly modify and extend the efficient routing mechanism. First, we design ESR to be a source routing scheme, where the computation of end-to-end paths and Bloom addresses is carried out exclusively by the servers. Second, ESR determines not only the shortest efficient paths between source and destination based on the weighted topology, but also derives the paths that are only reasonably longer than the shortest paths. The path length scaling factor (stretch) is denoted by γ. Based on γ, ESR calculates multiple paths between the end-points; some paths may have an additional hop compared to the shortest efficient paths, but this does not affect routing performance noticeably. This extension assures that the ESR method derives multiple paths even if the data center is built out of identical switches. Finally, two paths are picked randomly from the feasible paths: one for routing the packets of the flow and one as a backup that can be used immediately in case of failures. All packets of a given flow are routed on the same path; however, the load of multiple flows is spread on multiple paths. Accordingly, ESR achieves adequate load-balancing in Scafida data centers. The ESR algorithm is defined in Figure 4. Note that the size of routing tables is manageable, as each server is only required to store entries for other servers to which an active flow exists.

A recent measurement study [6] revealed that data center traffic is dominated by very short flows. In such a dynamic environment, a source routing scheme which probes the network for good end-to-end paths when a new flow is initiated (e.g., BCube's BSR [13]) is ineffective; the results of such probes are probably not valid by the time packets start to traverse the chosen route. ESR does not require any kind of signaling during normal operation. Instead, ESR relies on its load-balancing enabled path computation, and achieves good results. As a bonus, commodity switches do not have to respond to probes, hence saving valuable resources.

```
Input:
  $G(V,E)$ — the Scafida network
  $\beta$,$\gamma$ — parameters
  $d_v$ — degree of node $v$
  $s$ — the source of the demand
  $t$ — the target of the demand

Algorithm
1  for $e = (n_1, n_2) \in E$ do
2      $w_e = d_{n_1}^{\beta}$ // weight of the link
3  sp = shortestPath$(s, t, w)$
4  $c$ = cost of sp
5  feasiblePaths = []
6  for $\forall p$ between $s$ and $t$ do
7      cp = cost of $p$
8      if $cp \leq c \cdot \gamma$ then
9          feasiblePaths ← $p$
10 ep = random(feasiblePaths), bp = random(feasiblePaths $\setminus \{ep\}$)
```

Fig. 4. Path computation in Efficient Source Routing

Example. We illustrate the Efficient Source Routing method in Figure 5; only links of interest are shown (actual switch degrees in parentheses). Server A wants to send packets to server B; the ESR method is run at server A to determine the path on which the packets will traverse. Let us suppose that the ESR is operated with $\beta = 1$ and $\gamma = 1.1$. In this case, the length of the upper path is $16 + 4 + 4 + 16 = 40$ while the value of lower path is $16 + 24 + 16 = 56$. Accordingly, the upper path is the only feasible path as the lower path is longer than $40 \cdot 1.1 = 44$. By selecting the upper path, the flow avoids those links where high utilization is more likely as they are connected to switches with higher degrees. The impact of γ can be illustrated if $\gamma = 1.4$ is used, in this case both paths are feasible and therefore one is picked randomly. On the contrary, in case of $\beta = 0$ the lower path is selected as it has only four hops as opposed to the five hops of the upper path.

Assume the latter scenario, hence the path $A \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow B$ is chosen by the path computation procedure. Next, the Bloom IDs of the affected links are aggregated into a single Bloom address of 011011 (boxed). The packet first arrives at switch 3, where the in-packet Bloom filter only matches the link that goes towards switch 4. Similarly, the Bloom address of the packet is compared to the link IDs at switch 4. As the packet only matches the 01000 pattern (other links not shown), it is forwarded towards switch 5. There, the address matches the link ID towards server B, therefore, the packet follows that link, and arrives at server B. Now, if server A wants to send a multicast packet to both Server B and C, it has to compute the in-packet Bloom filter accordingly. Let us assume that the path computation procedure selects the same path towards server B, and $A \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow C$ for server

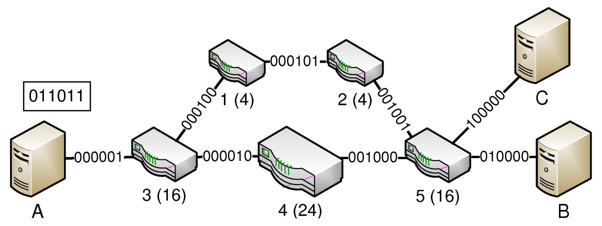


Fig. 5. Illustration of the Efficient Source Routing mechanism; switches are numbered (degree in parentheses) and Bloom link IDs are shown

C. As the link $5 \rightarrow C$ has the Bloom ID of 10000, the resulting Bloom address will be 111011. This address will match both links depicted at switch 5; thus, the packet is delivered to both destination servers.

C. Failure Handling

The Scafida structure inherently tolerates network failures owed to its scale-free network inspired design as noted in Section II. At the routing level, network failures are handled as follows. Failure of a network link is detected by the two neighboring nodes based on the loss of connection at the link layer. Failure of a switch/server is detected as multiple link failures by all neighboring nodes. Upon detecting a failure, a node generates a network failure (NF) control packet and sends it on all of its active links. The control message is propagated in the network based on a pruned broadcast method; accordingly, all servers will be aware of the network failure. The active paths that are affected by the failure are switched to their backup paths, while new flows are routed based on the updated network topology using the ESR method.

The NF control message has a Bloom address of $11\ldots1$; thus, it will match every possible Bloom ID at the switches. The source of the packet is set as the Bloom ID of the unavailable network link. The source addresses of the control messages are stored at every switch for a given, short time period (as a soft-state). An incoming NF packet is forwarded only if it is not present in the failed links table; otherwise, the packet is dropped. On the other hand, if a network failure is corrected, a network recovery (NR) control packet is broadcasted in the network; consequently, notifying the servers about the availability of the restored network equipment.

The proposed failure handling method does not have a significant impact on the performance of Scafida data centers: the generated control traffic overhead is proportional to the number of failures in the network. Based on [12], the number of simultaneous network failures is usually below a few hundred in very large data centers; thus, the aggregate load of the failure control messages is negligible compared to the throughput capability of the data center networks.

D. Routing Performance

We have evaluated the proposed ESR mechanism in a discrete event simulator. Traffic was simulated at the flow-level, assuming fair sharing of link bandwidth among competing flows. Here, we investigate two different scenarios: the impact of ESR weighting parameter β and ESR fault tolerance.

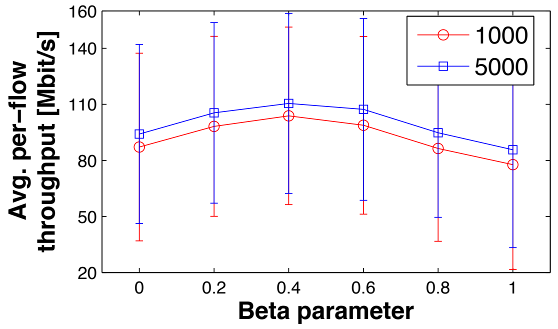


Fig. 6. The effect of the weighting factor β on per-flow throughput

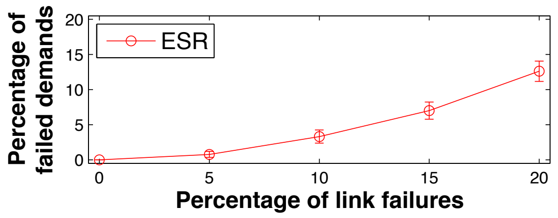


Fig. 7. Flow abortion ratio as the function of network failures

First, we created 1000- and 5000-server Scafida topologies with $m = 2$, $p_{t_0} = 2$ and switch size distribution of (60,50,40,30,17) for the 1000-server and (300,250,200,150,85) for the 5000-server topology, with switches having 4, 8, 16, 24, or 48 ports. We simulated 1000 flows with exponentially distributed inter-arrival times ($\lambda = 1000$ to ensure significant competition among flows) and lognormal (ln N(10,2) from [6]) flow size distribution. The 1000 flows were destined at a randomly chosen group of 100 servers to induce meaningful cross-traffic. Link capacities were uniformly set to 1 Gbit/s, and ESR's stretch bound was $\gamma = 1.1$. Results in Figure 6 are averaged over 20 simulation runs per parameter setting. β has a profound effect on per-flow throughput: the $\beta = 0$ case is shortest path routing, and it is clearly inferior to efficient routing with some larger weighting parameter. Also, too large β values will result in congestion, hence lower per-flow throughput. Note that the throughput values are higher in the larger topology because of the increased number of possible efficient paths.

Second, we investigated the fault tolerance properties of ESR ($\lambda = 500$ for inter-arrival times, $\beta = 0.5$, $\gamma = 1.3$). Results in Figure 7 show that ESR reacts well to network equipment failures, resulting in almost 90% of flows completed even at a 20% link failure ratio. We can conclude that ESR utilizes the inherent fault tolerance of the underlying Scafida topology in an efficient manner.

V. Implementation and Validation

As a proof of concept we implemented a prototype of Scafida in OpenFlow [23]; the implementation is available at [33]. Note that we chose OpenFlow because we are familiar with it, however, no part of the architecture is specific to OpenFlow. Besides illustrating the basic functionality of Scafida in the followings we briefly describe the implementation of several advanced features like multipath routing, multicasting, and failure discovery.

A. Main operation

For bootstrapping the system we use the NOX controller [23] to configure all the switches in the network. The join of a switch to the controller triggers the downloading of the configuration data in the form of permanent wildcarded flow entries dedicated to each port. A single flow entry contains a Bloom ID in the destination MAC address field and an *output action* to the appertaining port, while all other fields are wildcarded. If a given port connects the switch to a host, a *MAC address modifying action* is initiated to ensure packet delivery to the host. By the end of the configuration process flow tables of switches contain as many entries as they have ports containing the appropriate Bloom ID to port mappings.

After the switches are configured, the Scafida network is up and running and ready to route packets between the host machines. On each host, we run a Scafida daemon keeping track of the correct Bloom filter encodings of the paths towards all possible destinations. This daemon is aware of the actual topology, and it sets the ARP cache of the host to contain the destination IP to Bloom filter mappings. Therefore, upon sending a packet the host consults its ARP cache and retrieves the appropriate Bloom filter as the MAC address of the destination. In each switch, the Bloom filter matches only one[1] entry that indicates the appropriate output port. The last switch on the route replaces the destination MAC address of the packet with the MAC address of the host, and the packet gets delivered. The backward process works the same way. Note that the NOX controller is not involved in the forwarding operations.

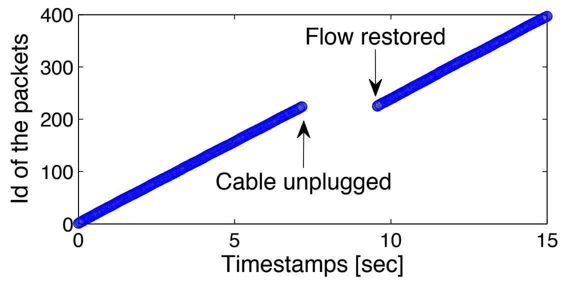
This type of operation also makes it easy to introduce flow-level multi-path communication. The ESR routing implemented in the Scafida daemon updates the ARP cache frequently, so the incoming flows retrieve different Bloom filters for the same IP address from the cache, thereby communicating on multiple paths with the same host.

B. Multicast

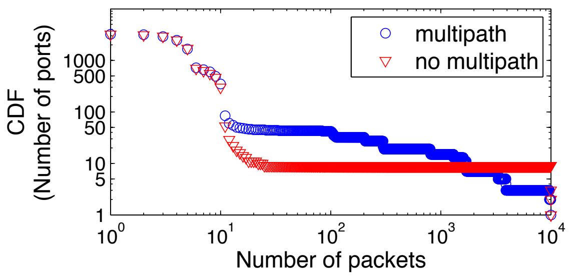
The application of Bloom filters makes it possible to implement multicast in a simple way. If a Bloom filter matches multiple flow table entries in a switch then forwarding on all matching output ports naturally realizes multicast. Since OpenFlow supports routing based only on the first matching entry in the flow table we extended the implementation of Open vSwitch [27] to be able to seek for multiple matches in the flow table. In order to have both multicast and the high performance of first match based forwarding, we mark the multicast packets in the header, and multiple matching is triggered exclusively when receiving a multicast header. This way the processing of a unicast packet happens pretty much the same way as before, maintaining high performance unicast forwarding.

The implementation of the multiple matching process relies on processing lists of matched rules. When a packet is received by the switch, it scans through the complete classifier flow table for matching entries. If there is a match, the associated classifier rule is concatenated to a list (this list is empty in the beginning of the process). When the scan passes the last entry in the flow table, the multiple matching process is finished, and the actions associated to the matched rules can be put together in a meta-rule to be performed. At the end, the meta-rule is installed on the data path.

[1]Note that we choose false-positive free routes exclusively.



(a) Failure discovery on Scafida testbed



(b) Multipath efficiency

Fig. 8. Measurements

C. Failure discovery

Discovering link failures and notifying end hosts are basic processes in Scafida to calculate the source route Bloom addresses in an up-to-date manner. To discover link failures the Scafida prototype uses heartbeat messages and timers according to the standard Link Layer Discovery Protocol (LLDP) [19]. If a switch does not receive an LLDP packet on a given port before its timer expires, then it assumes a link failure and triggers an intelligent flooding of this information. If an LLDP packet is received through a port for which the timer is previously expired, then a link recovery is assumed which also eventuates pruned flooding. The switch that receives the link failure/recovery message creates a non-permanent wildcarded flow entry containing the Bloom ID of the appertaining port in the source MAC address field, wildcards all fields except the source MAC address and associates a *drop* action with it. This way, upon receiving this notification multiple times, the message is dropped. Unfortunately, OpenFlow implements LLDP currently only in the controller, therefore, this feature scales poorly in the prototype.

D. Measurements

To validate and evaluate the implementation, we have established a 19 node Scafida network in a laboratory environment. The testbed consists of two types of commodity switches, namely 2 pieces of a 24-port HP ProCurve 6600, 7 pieces of TP-Link TL-WR841ND 4+1 port switches, and lab PCs. The commodity switches need special firmware to support the OpenFlow protocol. In case of HP switches, we use the v2.02s (K.14.77) OpenFlow-capable firmware while the TP-Link devices are operated by OpenWRT [28] Backfire 10.03.1-rc4 with the OpenFlow extension. [2] We use NOX v0.8.0 OpenFlow controller operating in out-of-band control mode in a separate management network.

Next, we demonstrate the failure discovery mechanism of the network. We establish a single flow between two servers and after a while unplug a cable on the communication route. In Figure 8(a), the IDs of the received packets are plotted against the timestamps at the receiver side. The link failure is detected by the system and the information is propagated to the end hosts, thus, the sender changes the Bloom address to send the packets on a different path. The recovery mechanism is performed within 4 sec in this scenario and the transmission is continued. This behavior is affected by the time parameters of the discovery application of NOX, as well, and inadequate parameters can lead to oscillation of the network. [3] We emphasize that the efficiency and scalability of the mechanism can be enhanced by implementing LLDP in the switches.

To investigate multipath efficiency we run a measurement on a 484-node Mininet [21] network, an emulated OpenFlow testbed. We started 100 flows with 100 packets each between two random hosts of the DC, which traversed on diverse routes according to ESR. Figure 8(b) shows the CDF of the tx/rx statistics of the ports over all switches in the network compared to the case when no multipath is issued. One can see that the 10000 packets travel along the same path when there is no multi-path capability, which eventuates large rx/tx values, while ESR effectively distributes the load among numerous ports in the network.

VI. Cost considerations

Simulation results and experimental evaluation show that our approach satisfies its design goals and is worth considering in production networks, especially for SMEs and NPOs with ad-hoc network expansion opportunities and on a strict budget. With that said, there are certain areas of expenditures for deeper consideration.

Customizing for a set budget. Throughout the paper we assumed a fixed set of available commodity hardware, over which a data center should be built. While this is a plausible scenario, an organization can also have a fixed budget to spend on hardware and zero available equipment. Upon gathering pricing information, our simulation tools could be used to choose an optimal set of switches and servers resulting in the best performing Scafida data center. A recent cost comparison of several data center architectures discussed cases where special topologies are preferred from an expenditure point of view [29]. Due to its flexible structure and competitive structural properties, Scafida can be used as a single data center architecture in all the cases. Thus, Scafida allows stakeholders to decide about the aggregate expenditure of the data center instead of selecting a specific topology based on assumed future market scenarios (e.g., the price of switches vs. servers).

Cabling. Our experience with small enterprise and university networks suggests that usually very little emphasis is put on proper and clearly structured racking and cabling. While ad-hoc solutions can certainly work on a small scale, multiple thousands of data center nodes would certainly benefit from order. In our case, this implies finding subgraphs in the topology which can be stored in the same rack. This constitutes important future work for us. From a cost viewpoint, connecting the equipments of a Scafida DC would require intense human workforce; however, as the absolute cost of cabling is marginal compared to the aggregate cost of DC architectures [29], Scafida deployment would not be hindered by cabling costs.

Power consumption. Scafida data centers have the potential to be energy efficient due to their scalable and flexible structure. As structures of any size can be realized using Scafida, the resulting data center is energy proportional; i.e., the power consumption is proportional to the number of servers. On the contrary, the power consumption of state-of-the-art data center architectures is not energy proportional [15, 17]. Current energy price trends predict that energy proportionality will be a crucial property regarding DC expenditures as the price of power tends to increase. On top of this, a network-wide power manager such as ElasticTree [18] can further be used inside Scafida, switching off unnecessary servers and network elements in the network without sacrificing throughput.

Commodity switches. Scafida is designed to create DCs out of commodity switches using a flexible topology. Scafida has a specialized routing method to exploit the benefits of its topology; however, this does not increase the cost of its deployment. The firmware of of-the-shelf network equipments can be easily modified [28]; thus, a Scafida DC can be created using cheap commodity switches. For example, the cost of the devices of our prototype is extremely low: the TP-Link switches cost $34 according to Amazon but also the price of the 24-port HP switches is moderate ($3800).

VII. Related Work

There is a sizable and highly practical body of work related to data center networking. Several data center architectures have been proposed recently; most of them are based on symmetric structures. Data centers based on fat-tree topologies [2, 12, 26] are built using commodity switches arranged in three layers, namely core, medium, and server layers. The structure is also known as Clos topology. The LEGUP proposal tackles the problem of efficiently expanding existing data centers, adding heterogeneity to Clos networks [8].

The symmetric structure of the BCube [13] data center architecture is designed using a recursive algorithm. BCube is intended to be used in container based, modular data centers, with a few thousand servers. The MDCube architecture proposes a method to interconnect these containers to create a mega-data center [34]. Similarly to BCube, DCell is also

[2]This model of TP-Link requires a slight modification in the kernel driver of the switch, more exactly the MAC learning function has to be disabled in the corresponding source file (`ag71xx_ar7240.c`).

[3]More specifically, `TIMEOUT_CHECK_PERIOD` and `LINK_TIMEOUT` parameters are set 1 and 4.5 sec, respectively.

built recursively [14]. DCell's recursive expression scales up rapidly, thus, DCell can have enormous number of servers with small structural levels and switch ports.

Portland [26] is a scalable, fault tolerant layer-2 data center fabric built on multi-rooted tree topologies (including fat tree as well) that supports easy migration of virtual machines. VL2 [12] is an agile data center fabric with properties like performance isolation between services, load balancing, and flat addressing.

Jellyfish [32], a recent DC architecture proposal suggests to use random networks as underlying DC topology. Due to the random structure, incremental expandability of this DC is solvable; however, the routing in Jellyfish DCs is identified as an open research challenge. Despite the similarities, Scafida has an advantage over Jellyfish owed to its structured randomness inherited from scale-free networks: routing inside a Scafida DC can be done throughout our proposed ESR method.

A recent study provides key insights from production data centers [6]. Among others, it shows that data center traffic is dominated by short and small flows, predominantly short flow inter-arrival times require quick forwarding decision in switches, and centralized fabric managers need to be implemented in hardware to scale with growing network size.

Various studies dealt with the properties of scale-free (or complex) networks in the recent decade, as a starting point please refer to [4].

VIII. Conclusion

We presented the design, implementation, and multi-faceted evaluation of the Scafida data center network architecture. Scafida's basic structure consists of servers and commodity switches, connected into a topology inspired by scale-free networks as introduced in [16]. In this paper we proposed an organic expansion algorithm for Scafida resulting in a freely scalable data center network structure with favorable incremental properties (scalability and expandability) and performance characteristics (short paths, high bisection bandwidth and fault-tolerance). On top of its interconnection structure, Scafida runs a fault-tolerant source routing protocol, ESR, performing low-stretch, load-balanced, multi-path, multicast-enabled routing, while requiring minimal computation in switches. We have also implemented an OpenFlow-based Scafida prototype, and validated its main features via Mininet-emulations and live testbed measurements. We made the source code of our prototype implementation, simulator, and additional evaluation results available at [33]. We believe that both small and large, continuously expanding data centers could benefit from our findings.

References

[1] Advanced Systems Group. Private vs. public cloud financial analysis. *http://www.virtual.com/solutions/cloud-computing/cloud-financial-analysis*.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM '08*, pages 63–74, 2008.

[3] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–82, August 2000.

[4] A. Barabási. Scale-free networks: a decade and beyond. *Science*, 325(5939):412, 2009.

[5] A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.

[6] T. Benson, A. Akella, and D. Maltz. Network Traffic Characteristics of Data Centers in the Wild. *IMC '10*, pages 267–280, 2010.

[7] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[8] A. R. Curtis, S. Keshav, and A. Lopez-Ortiz. Legup: using heterogeneity to reduce the cost of data center network upgrades. In *Co-NEXT '10*, pages 14:1–14:12, New York, NY, USA, 2010. ACM.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI'04*, San Francisco, 2004.

[10] Defense Systems. Dod tackles information security in the cloud. *http://defensesystems.com/articles/2011/01/24/defense-it-1-dod-cloud-computing-security-issues.aspx*.

[11] Forrester Research. Market overview: Private cloud solutions, q2 2011. *http://www.forrester.com/rb/Research/market_overview_private_cloud_solutions%2C_q2_2011/q/id/58924/t/2*.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *SIGCOMM '09*, pages 51–62, 2009.

[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM '09*, pages 63–74, 2009.

[14] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *SIGCOMM '08*, pages 75–86, 2008.

[15] L. Gyarmati and A. Trinh. Energy efficiency of data centers. In *Green IT: Technologies and Applications*, pages 229–244. Springer-Verlag, 2011.

[16] L. Gyarmati and T. A. Trinh. Scafida: a scale-free network inspired data center architecture. *SIGCOMM Comput. Commun. Rev.*, 40:4–12.

[17] L. Gyarmati and T. A. Trinh. How can architecture help to reduce energy consumption in data center networking? In *e-Energy '10*, pages 183–186, New York, NY, USA, 2010. ACM.

[18] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *NDSI '10*, 2010.

[19] IEEE 802.1AB. Station and media access control connectivity discovery.

[20] R. Katz. Tech titans building boom. *Spectrum, IEEE*, 46(2):40 –54, 2009.

[21] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *HotNets '10*, page 19. ACM, 2010.

[22] A. Licis. HP, data center planning, design and optimization: a global perspective. http://goo.gl/Sfydq, June 2008.

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[24] R. Miller. Facebook now has 30,000 servers. http://goo.gl/EGD2D, Nov. 2009.

[25] R. Miller. Facebook server count: 60,000 or more. http://goo.gl/79J4, June 2010.

[26] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09*, pages 39–50, 2009.

[27] Open vSwitch. http://openvswitch.org/.

[28] OpenWrt. http://openwrt.org/.

[29] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica. A cost comparison of datacenter network architectures. Co-NEXT '10, pages 16:1–16:12, New York, NY, USA, 2010. ACM.

[30] C. Rothenberg, C. Macapuna, F. Verdi, M. Magalhães, and A. Zahemszky. Data center networking with in-packet Bloom filters. In *SBRC, Gramado, Brazil*, 2010.

[31] M. Sarela, C. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, and J. Ott. Forwarding anomalies in bloom filter-based multicast. In *INFOCOM, 2011 Proceedings IEEE*, april 2011.

[32] A. Singla, C. Hong, L. Popa, and P. Godfrey. Jellyfish: Networking data centers, randomly. In *USENIX HotCloud'11*, 2011.

[33] Supporting materials. https://sites.google.com/site/infocom2012freescalingdc/.

[34] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang. Mdcube: a high performance network structure for modular data center interconnection. In *CoNEXT '09*, pages 25–36, New York, NY, USA, 2009. ACM.

[35] G. Yan, T. Zhou, B. Hu, Z. Fu, and B. Wang. Efficient routing on complex networks. *Physical Review E*, 73(4):46108, 2006.