# Poincaré: a Hyperbolic Data Center Architecture

Márton Csernai, András Gulyás, Attila Kőrösi, Balázs Sonkoly
Budapest University of Technology and Economics
Department of Telecommunications and Media Informatics
Email: {csernai,gulyas,korosi,sonkoly}@tmit.bme.hu

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

***Abstract*—Current trends in cloud computing suggest that both large, public clouds and small, private clouds will proliferate in the near future. Operational requirements, such as high bandwidth, dependability and smooth manageability, are similar for both types of clouds and their underlying data center architecture. Such requirements can be satisfied with utilizing fully distributed, low-overhead mechanisms at the algorithm level, and an efficient layer 2 implementation at the practical level. On the other hand, owners of evolving private data centers are in dire need of an incrementally upgradeable architecture which supports a small roll-out and continuous expansion in small quanta. In order to satisfy both requirements, we propose *Poincaré*, a data center architecture inspired by hyperbolic tessellations, which utilizes low-overhead, greedy routing. On one hand, *Poincaré* scales to support large data centers with low diameter, high bisection bandwidth, inherent multipath and multicast capabilities, and efficient error recovery. On the other hand, *Poincaré* supports incremental plug & play upgradability with regard to both servers and switches. We evaluate *Poincaré* using analysis, extensive simulations and a prototype implementation.**

***Index Terms*—data center, incremental upgrade, hyperbolic tessellation, greedy routing, distributed operation, scalability**

## I. Introduction

Cloud computing has been emerging to be the dominant operation model of present and future networked services. In order to provide the underlying pervasive networking functions, data centers have to scale up to previously unseen proportions. Tens of thousands of servers is already the norm for large providers, and this number is predicted to grow significantly over the next few years, as services, storage, as well as enterprises and users are increasingly relying on the cloud. Factor in virtualization and we are easily in the range of a million virtual end points. In parallel to migrating to huge, public clouds, another trend is gaining momentum: more and more organizations decide to consolidate their computing resources into small- or medium-scale private clouds [1]. 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 [2]. Second, private cloud operation policies and management procedures can be tailor-made to the owner's liking [1]. Finally, of course, cost is always a deciding factor; surprisingly, operating a private cloud could be advantageous in the long(er) run [3]. As a consequence, the increasing proliferation of both small and large data centers are highly likely.

There are two orthogonal requirements for a data center design to suit both the needs of small-and-starting and large-and-evolving data centers. First, such a design should be *structurally upgradable*: a small company should be able to start a private data center quickly and with a limited budget, and build it out incrementally in small quanta [4]. 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 [5], resulting in a doubling of servers over the course of 7 months in 2010 [6]. With most designs providing mechanisms only for large-scale expansion [7] [8], little work has been already done regarding incremental upgradability in data centers. LEGUP [9] leverages hardware heterogeneity to reduce the cost of upgrades in fat-tree based data centers. Jellyfish [10] and REWIRE [11] propose a non-symmetric topology to facilitate upgradability of data centers; however, they sacrifice structure creating a significant challenge for their future routing mechanism and performance upgrades.

Second, the respective data center design should be *operationally scalable* providing performance, dependability and manageability for tens or hundreds of thousands of network nodes. On the network algorithm level, this implies distributed, low-overhead mechanisms both for routing and failure handling. On the implementation level, due to both virtualization and network management reasons, data centers are often managed as a single logical layer 2 fabric. On the other hand, traditional LAN technologies do not scale well to the size of large data centers. Network layer techniques, such as large forwarding tables, loop-free routing, quick failure detection and propagation, etc. are needed to be realized utilizing only switch hardware. Recently proposed solutions, such as TRILL [12], SEATTLE [13] and Portland [14], address these issues to a certain extent, but they come with a cost of significant complexity at the switch/fabric control plane. Other architectures, like BCube [7] and VL2 [15], rely on servers routing to overcome this limitation. While certainly a worthwhile approach, with the advent of computation-intensive cloud services and the migration of vast online storages into the cloud, coupled with the cloud providers' economic incentives

to run their servers close to their full capacity [16], we argue that servers may face resource constraints if being the main responsible also for routing.

In this paper we propose *Poincaré*, a data center architecture which is by design structurally upgradeable and operationally scalable. The topology of *Poincaré* is inspired by hyperbolic tessellations, providing incremental expandability and favorable performance characteristics. *Poincaré* uses a fully distributed, low-overhead, greedy routing algorithm efficiently utilizing the features of the topology. Such a lightweight routing mechanism can be fully implemented in layer 2, while keeping both the control and the forwarding plane simple at the same time, enabling a data center built out of cheap network equipment. The main benefits of *Poincaré* are threefold. First, *Poincaré*'s hyperbolic structure provides analytically provable low network diameter and high performance greedy routing, multiple short paths between arbitrary nodes and high bisection bandwidth. Second, *Poincaré*'s greedy routing mechanism harness the aforementioned structural characteristics while allowing for natural local failure handling within failure detection time and low-overhead multipath and multicast routing. Finally, *Poincaré* supports incremental plug & play upgradability with regard to both servers and switches, while enabling a small initial rollout and a flexible, budget-conscious way of data center expansion. We justify our design with analytical proofs and extensive simulations augmented by a prototype implementation and testbed experiments.

The rest of this paper is structured as follows. Section II introduces the *Poincaré* data center structure based on hyperbolic tessellations. In Section III we present our greedy geographic routing algorithm is along with its multipath, multicast and error recovery extensions. The incremental structural and performance upgrade process is carefully described in Section IV. In Section V we thoroughly evaluate the performance of *Poincaré* via simulation. Section VI introduces the *Poincaré* prototype implementation and provides testbed experiment results. Section VII presents practical considerations on cabling, initial rollout and expansion costs. Finally, related work is described briefly in Section VIII and the paper is concluded in Section IX.

## II. Structure: A Trip to the Hyperbolic Space

A tree is a very cost effective interconnection structure when routing has to be solved on a population of network nodes. A $k$-ary tree can provide low diameter (low delay), low average degree (low cost), easy loop-detection and simple routing decision in the nodes [17]. Such compelling properties qualify trees to be utilized in an array of routing protocols (STP, OSPF, ISIS) and, more recently, in data centers. On the negative side, trees cannot ensure path diversity and high throughput: two key requirements for data centers architectures. Recently, several augmentations of trees have been proposed to overcome these limitations by densification (e.g., Clos networks) providing multiple paths, large bisection bandwidth and no single point of failure [18] [15]. A common drawback of such approaches is that when expanding the DC these "embedded" interconnection structures has to be carefully maintained and sometimes completely replaced and rewired[1] to keep up with the number of servers. Such complete rewiring in fact mean the building of a new DC from scratch. LEGUP [9] addresses this issue by allowing for heterogeneous switches; yet, it is only suited for large upgrades and relies on higher layer mechanisms for performance.

Since we require *Poincaré* to be incrementally upgradeable total rewiring is unacceptable. Thus we must need an architecture which can *preserve its inherent structure* regardless of the number of servers and still provide path diversity and high throughput. In the following we design a topology inspired by tessellations embedded in the hyperbolic plane exhibiting structural similarity with trees [19]. A tessellation can be interpreted as an augmentation of the tree structure in which the branches are connected and can exchange traffic without affecting the core.

### A. The basic topology of Poincaré: hyperbolic tessellations

As a minimum underlying topology *Poincaré* contains a regular tessellation of the hyperbolic plane in the Poincaré disk model [20]. In this model the points of the hyperbolic plane are mapped to the unit disk. The $(n, k)$ regular tessellation[2] uses regular $n$-gons from which $k$ meet in a given vertex to fill the hyperbolic plane with no overlaps and no gaps. If we consider the vertices of the polygons as nodes, and the sides as links, we have a regular topology embedded into the hyperbolic plane. Figure 1 shows a fat-tree topology and a $(5, 4)$ tessellation with similar topological parameters. As it will be shown *Poincaré*'s routing can effectively utilize the "side" links to exchange traffic between "regions" of a tessellation without affecting the core. This property enables us to use a fixed size core (as opposed to fat-tree's variable sized core when upgrading to larger core switches) and implement incremental server and performance upgrades. In the figure we plotted a traffic heat map of an all-to-all traffic scenario for a 15 node fat-tree topology and a basic *Poincaré* topology with similar topological parameters (average degree, average path length).

The coordinates of the nodes for a given $(n, k)$ tessellation can be easily computed by implementing simple geometric mappings in the hyperbolic plane (see for example [21] for Java code) for arbitrary network size. The derived coordinates will point to the logical places of servers and switches, and also the link positions. Using the tessellation as a map we can grow *Poincaré* by placing servers and switches to the unoccupied node positions starting from the inner part of the tessellation. The detailed growth algorithm is introduced in Section IV.

Such a tessellation topology immediately possesses compelling properties. In the sequel we prove that the diameter of

[1] When the arity of the tree has to be increased.

[2] As opposed to Euclidean tiling there exist a $(n, k)$ tiling on the hyperbolic plane if $\frac{1}{n} + \frac{1}{k} < \frac{1}{2}$ because on the hyperbolic plane the angles of a regular n-gon could be arbitrary small

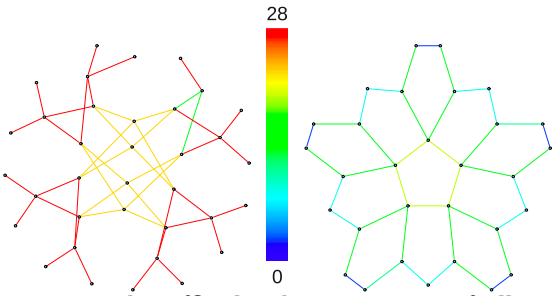



Fig. 1: Link traffic distribution in case of all-to-all traffic in (left) fat-tree and (right) tessellation structures.

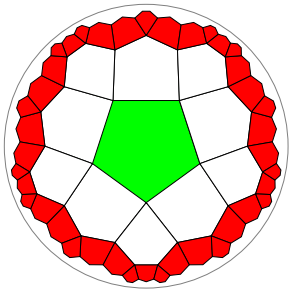


Fig. 2: The layers of the recursion is shown with different colors.

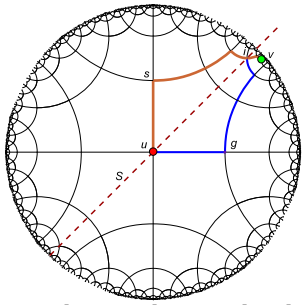



Fig. 3: The next-hop on the shortest path is also a greedy next-hop.

such a network grows logarithmically with the network size while the degree of the nodes is bounded by $k$.

*Theorem 1:* The diameter of the $(n, k)$ tessellation grows logarithmically with the number of nodes.

*Proof:* Denote the number of polygons and vertices on the "perimeter" of the $m$-th level with $p_m$ and $v_m$ respectively (see Fig. 2). For any $n > 3$, $p_{m+1}$ and $v_{m+1}$ can be calculated from $p_m$ and $v_m$ according to the following recursion.

$$\begin{aligned} v_{m+1} &= (n-3)v_m + (k-3)(n-2)v_m - (n-2)p_m \\ p_{m+1} &= v_m + (k-3)v_m - p_m \end{aligned}$$

Solving the recursion we get:

$$\begin{pmatrix} v_m \\ p_m \end{pmatrix} = \begin{pmatrix} (k-2)(n-2)-1 & -(n-2) \\ k-2 & -1 \end{pmatrix}^m \begin{pmatrix} v_0 \\ p_0 \end{pmatrix}$$

Let $M$ be the matrix of the recursion, then $M^m$ can be written as $M^m = Q\Lambda^m Q^{-1}$, where $Q$ is the matrix of the eigenvectors of $M$ and $\Lambda$ is a diagonal matrix whose nonzero elements are the corresponding eigenvalues. In our case

$$\begin{aligned} Q &= \begin{pmatrix} -\frac{\sqrt{(Z-2)^2-4}-Z}{2(k-2)} & -\frac{-Z-\sqrt{(Z-2)^2-4}}{2(k-2)} \\ 1 & 1 \end{pmatrix} \\ \Lambda &= \begin{pmatrix} \frac{Z-2-\sqrt{(Z-2)^2-4}}{2} & 0 \\ 0 & \frac{Z-2+\sqrt{(Z-2)^2-4}}{2} \end{pmatrix} \end{aligned}$$

where $Z = (n-2)(k-2)$. Hence by using $v_0 = n$ and $p_0 = 0$ we get

$$\begin{aligned} v_m &= n\frac{(Z-2-Y)^m(Y-Z) - (Z-2+Y)^m(Y+Z)}{2^{1+m}Y} \\ &\approx n(Z-1)^m = n\left((n-2)(k-2)-1\right)^m \end{aligned}$$

where $Y = \sqrt{(Z-2)^2-4}$ and the last step is due to $Z \approx Y$.

For the special case $n = 3$

$$v_{m+1} = (k-4)v_m - v_{m-1}.$$

Let $\lambda_{1,2}$ be the solutions of equation $r^2 - (k-4)r + 1 = 0$, then $v_m$ can be written in the following form:

$$v_m = a\lambda_1^m + b\lambda_2^m,$$

where $a$ and $b$ can be calculated from $v_1 = 3$ and $v_2 = 3(k-3)$.

Since the number of vertices grows exponentially with the number of levels, the diameter is at most $(2m+1)\frac{n}{2}$ which completes the proof. ■

### B. Performance topologies

To boost the performance of a given *Poincaré* topology we can add more links between our nodes without any restraints, hereby improving topological (diameter, average hop distance) and operational properties (error tolerance, multipath, multicast, throughput). For evaluation purposes we define the following simple algorithm to produce a family of graphs by adding more links: take a tessellation and a radius $r$, and connect the nodes whose distance in the hyperbolic space is less than $r$. This way link locality is preserved hereby moderating cabling costs. Table I presents how the characteristics of the topology is enhanced by adding more and more links for three types of 1000-node tessellations. We note that our simple algorithm increases the average degree by large quanta, but we emphasize that link additions can be done with arbitrary granularity and using diverse algorithms to make the best out of a fixed budget.

| | | r | Avg. deg. | Max. deg. | Avg. dist. | Diameter |
|---|---|---|---|---|---|---|
| 16-triangle | sparse | 3.5 | 4.09 | 16 | 5.20 | 7 |
| | medium | 4.8 | 6.27 | 32 | 4.29 | 6 |
| | dense | 5.5 | 9.26 | 64 | 3.70 | 5 |
| 6-pentagon | sparse | 3.3 | 5.96 | 20 | 5.14 | 7 |
| | medium | 4.2 | 8.15 | 34 | 4.26 | 6 |
| | dense | 4.4 | 10.9 | 51 | 3.79 | 5 |
| 4-decagon | sparse | 3.2 | 6.5 | 20 | 4.97 | 8 |
| | medium | 3.3 | 7.59 | 24 | 4.52 | 6 |
| | dense | 3.7 | 9.86 | 36 | 4.22 | 5 |

TABLE I: Tessellations with different (n, k) parameters.

### C. Comparison of DC topologies

Table II presents the properties of 4000 server *Poincaré* topologies with corresponding fat-tree and BCube structures. One can see that *Poincaré* is comparable with the widely used fat-tree and BCube topologies in terms of diameter, average path length and bisection bandwidth, while using less switches. Current technological trends allow us to depend on larger switches (in the range of 100-150 ports[3]) as the price of such switches grows only linearly with the number of their ports [22]. Also note that due to its heavy-tailed degree distribution *Poincaré* uses only a small number (10) of large switches as it can be seen on Fig. 4.

[3]VL2 for example uses at least 40-port switches for a 4000-server DC topology.

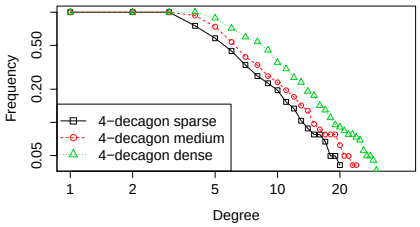


Fig. 4: Cumulative degree distribution of the different density 4-decagon topologies.

| DC Type | Switches | Diam. | Avg. dist. | Max. deg. | Bis. bw. |
|---|---|---|---|---|---|
| Fat-Tree($n = 28$) | 980 | 6 | 5.23 | 28 | 7471.9 |
| BCube($n$=16,$k$=2) | 762 | 8 | 5.36 | 16 | 6002.7 |
| *Poincaré* (4-decagon) | 640 | 9 | 4.85 | 76 | 7334.7 |

TABLE II: Topological comparison of different DC topologies.

## III. Routing: Greedy Geographic Routing

*Poincaré* routes greedily on the geographic coordinates of nodes in the tessellation. In this section we present the basic routing mechanism and prove its performance on *Poincaré*'s structure in an analytical manner. Simple yet efficient multi-path and multicast algorithms are also proposed, which both leverage and enhance the greedy routing paradigm, and sustain the low overhead operation manner.

### A. Basic mechanism

In greedy geographic routing, by default, the routing mechanism doesn't require routing states to be kept in the switching fabric. The routing decision is solely based on the metric distances between the link neighbor nodes $v(v_1, v_2)$ of $u$ and the destination node $t(t_1, t_2)$. An intermediate node on the forwarding path always forwards a packet to its neighbor closest to the destination. In *Poincaré* we use the distances between the nodes on the hyperbolic *Poincaré* disk as the metric for greedy routing:

$$d(v,t) = \text{arccosh}\left(1 + 2\frac{(v_1 - t_1)^2 + (v_2 - t_2)^2}{(1 - v_1^2 - v_2^2)(1 - t_1^2 - t_2^2)}\right)$$

If $u$ does not have any neighbor $v$ for which $d(v,t) \leq d(u,t)$ then greedy routing is in a local minimum and fails. Note that for example Bcube's single path routing algorithm[4] also suffers from such phenomena. To overcome this, Bcube's routing falls back to BSF search to be able to route when network failures are present. We will see that *Poincaré*'s structure ensures such events to happen only in case of massive network failures and with extremely small probability.

*Theorem 2:* Assuming no network failures greedy routing always finds optimal paths between arbitrary node pairs in an $(n, k)$ tessellation.

*Proof:* We prove this statement in an indirect manner. Assume that there exist node pairs between which the length of the shortest path is smaller than the greedy path. Among these pick the pair $(u, v)$ for which $h(u, v)$ is minimal, where $h(u, v)$ refers to the minimal hopcount between $u$ and $v$ in the tessellation. Figure 2 shows the tessellation from the point of $u$. Since $h(u, v)$ is minimal the greedy path must deviate from the shortest path at $u$. The red and the blue lines show the shortest and the greedy paths on which the first node is $s$ and $g$ respectively.

Due to the reflection symmetry of the tessellation there is an axis of symmetry $S$ which maps $s$ to $g$ and also separates[5] $s$ and $v$. Hence there exists at least one intersection $i$ of the shortest path and $S$. By symmetry $i$ can coincide with a node or can be a midpoint of an edge. Let's reflect the shortest path between $u$ and $i$ with respect to axis $S$. The mirror image and the part from $i$ to $v$ of the original shortest path is also a shortest path between $u$ and $v$. Since $g$ resides in the mirror image it is also a part of a shortest path. This is in contradiction with our assumption that $h(u, v)$ is minimal. ■

Although the proof above holds for an infinite hyperbolic tessellation only, our simulations readily showed that greedy routing can effectively find the optimal paths on finite and upgraded *Poincaré* topologies, thus eventuating quasi the same values for average distance and average greedy distance for the topologies shown in Table I.

Since arccosh() is monotone this operation can be left out of the calculation for boosting forwarding performance. Hence the required computation at each forwarding decision is reduced to about a dozen of simple arithmetic operations which consumes reasonably few CPU cycles. By default, nodes are required to calculate the next-hop distances for each packet to be forwarded by the intermediate nodes. As it is shown in Section VI in a working greedy routing environment, routes can be cached by intermediate switches, and flow labels can be used for per-flow forwarding decisions. We emphasize that *Poincaré* retains all advantages of greedy routing thus there is no link state propagation protocol prevalently used in DC architectures. This routing mechanism doesn't require carefully adjusted routing tables and implements routing with in essence zero messaging overhead.

### B. Low overhead greedy routing extensions

DC specific routing requires many features such as multicast (to support MapReduce induced traffic load and distributed storage systems) and multipath routing (for multipath TCP, error tolerance, VM migration). We extend *Poincaré*'s default greedy routing to support the above mentioned features in a way that maintains the low overhead and distributed routing operation.

**Multicast.** For *small multicast groups*, *Poincaré* implements an Xcast-like [23] mechanism where all destination addresses are stored in the header, and a packet is sent only once through a link for a given set of destinations. When common path towards all destinations cannot be maintained, the multicast group is split, the headers are rewritten and the packets are forwarded through different links accordingly. To reduce the volume of traffic, the routing mechanism in the nodes can be traced to the well known set cover problem, i.e., there is a multicast group (set), and we want to cover these destinations by as few outgoing links as possible. As opposed to Xcast

[4]A special kind of greedy routing in high dimensional space.

[5]This is simply because $g$ is closer to $v$ than $s$ also in Euclidean distance, while $u$ is in the center.

which can rely only on poor diversity of shortest paths, *Poincaré* leverages the abundance of greedy routes for a given multicast group. It means that there are numerous greedy routable next hop neighbors towards a given destination, and an intermediate node can chose a lower number of links to send a multicast packet out on. Our implementation employs a greedy heuristic to solve the set cover problem. Figure 5 shows the different path selection results for a source and a two-member destination group in case of Xcast (left) and greedy multicast (right). It can be seen that in case of greedy multicast the packet is sent on common path to both destinations as long as there is a common greedy routable link, hence saving precious link capacity. To speed up multicast forwarding in a working DC environment, the solution can make use of the recent implementation of the greedy heuristic function in NetFPGA architectures [24].

In current data center architectures *large multicast groups* are usually supported by introducing state to the switching fabric. Note that the space-embedded structure of *Poincaré* permits the addressing of larger groups by defining a *point* and an *offset* in the hyperbolic space. This type of addressing can designate servers whose distance from the point is less than the offset value, hereby succinctly addressing larger network segments. We leave the implementation of such a multicast solution for future work.

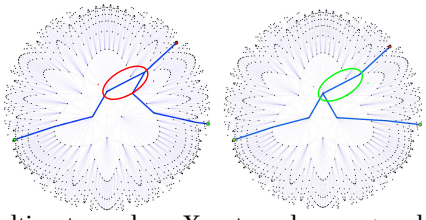


Fig. 5: Multicast modes: Xcast and our greedy multicast algorithm.

**Multipath.** For multipath purposes *Poincaré* uses the following simple distributed algorithm: for a new incoming flow, choose the least-loaded outgoing link through which the packets can reach the destination on a greedy path. Such a multipath algorithm relies strongly on the number of edge disjoint paths that can be used by greedy routing. To measure how many such paths exist between pairs of nodes in a *Poincaré* topology $G$, we generate a directed subgraph $G_d$ from $G$ for every $d$ destination. In $G_d$ a link pointing from $u$ to $v$ exists only if $u$ and $v$ are connected in $G$ and $d(u,d) > d(v,d)$. All link capacities are set to 1 and the maximal flow on $G_d$ is calculated. By applying the Max-flow min-cut theorem [25] we get the exact number of link-disjoint greedy routable multiple paths between $s$ and $d$. Table III shows the outcome of this process averaged for all source-destination pairs in 1000-node *Poincaré* topologies. After only moderate topology upgrades up to 4 greedy routable link disjoint paths are present in our simulated topologies. For easier positioning the results we note that the corresponding values for fat-tree, dual-homed fat-tree (DHFT) [26], and Bcube are 1, 2 and $k+1$ respectively, where $k$ stands for the number of Bcube levels.

| Topology | Avg. server ports (Max) | Avg. greedy disjoint paths (Max) |
|---|---|---|
| 4-decagon sparse | 3.371 (4) | 2.138 (4) |
| 4-decagon dense | 3.663 (4) | 2.569 (4) |

TABLE III: Number of greedy routable link disjoint paths.

### C. Failure handling

Greedy routing provides a fairly natural way of recovering from failures. Consider the scenario in Fig. 6 where PC1 sends traffic to PC2. From the coordinates we can compute the greedy path as PC2-Switch1-Switch2-PC2. If for example the link between Switch 1 and 2 goes down as indicated in the figure, Switch 1 notices the failure and for the next packet received from PC1 the greedy calculation gives Switch 4 as the next hop, hereby avoiding the failed link without any global failure propagation and route recomputation and requiring time only for the detection of the failure.

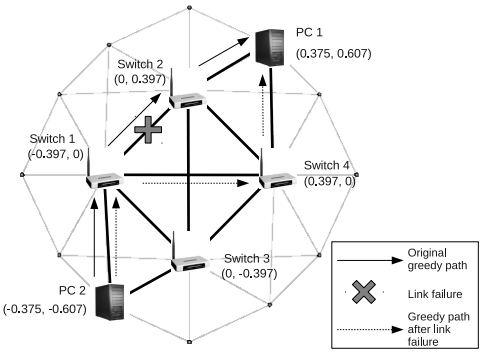



Fig. 6: (8,4) tessellation with coordinates and our testbed topology

In case of link failures it can occur that greedy routing fails (stuck in a local minimum) however there would be available non-greedy paths. We will show in Section V that the probability of such events to happen in a *Poincaré* topology is very low compared to e.g. the disconnection probability of servers from the fat-tree topology in case of link failures. Moreover we can exploit the path diversity of *Poincaré* to reduce the probability of greedy fails by considering the following algorithm. When a source node fails to find its destination with default greedy routing it can assign a random trajectory "bias" ($\alpha$) to the next packet and retry the transmission. Intermediate nodes use this bias to assigns weights $d_i^{\alpha}$ to their neighbors based on their distance $d_i$ to the destination and pick a neighbor with larger weight with greater probability. The effect of the different $\alpha$ parameter values on the greedy trajectories can be seen in Figure 7. By setting alpha to a very low value the algorithm always favors the shortest greedy path, while if set to a higher positive value, the traversed routes will be distributed among the many greedy routable paths hereby avoiding the local minimum. To completely eliminate this effect one can use recent improvements of greedy routing [27] [28], however such techniques notably increase routing complexity.

## IV. Structural growth and performance upgrades

This section discusses how *Poincaré* manages to satisfy the incremental upgradability requirements and describes various

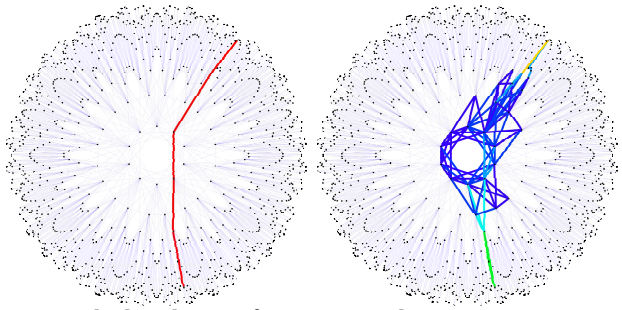


Fig. 7: Path distribution for a source-destination pair in case of single path $\alpha$ = -100 (left) and $\alpha$ = 1 (right).

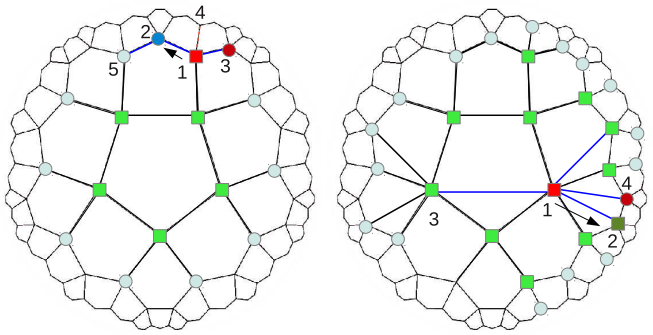
methods for capacity provisioning in a cost optimized manner. The upgrade process ensures that *any upgrade is a feasible upgrade*, while affecting only the immediate vicinity of the newly connected servers or switches, i.e., without impacting the data center core and main operation.

The tessellation based structure can be started from an arbitrary number of servers and switches, and can be gradually built out by adding more servers to the perimeter. The next layer of the recursive tessellation should be computed which determines the *possible* places where new servers can be added. We demonstrate the initial build out method in the following simple example. Figure 8(a) shows an initial 4-pentagon tessellation based structure. To start with, we have 5 4-port switches, and 10 servers. All switch ports are connected to either servers or other switches. When adding one additional server to the topology, we can shift a server node to an outer coordinate, put a new switch in its place, and then connect both servers to the new switch.

The coordinates of the devices can be computed and stored in a directory service to be used by manual configuration at new server or switch installment. Also an automatic coordinate assignment protocol could be implemented, which would tell newly installed devices their new coordinates based on the coordinates of their neighboring nodes. Besides this coordinate assignment no extra configuration is needed to get *Poincaré* up and running.

As demand is getting higher over time, the performance of *Poincaré* can be structurally upgraded by adding more switching equipment. Larger switches can replace smaller switches in a plug-and-play manner always resulting in better structural benchmarks. Naturally, adding links to the inner part of the topology can have larger impact. During the upgrade process, *Poincaré*'s multipath capabilities ensure the uninterrupted operation and evolution of the data center. When replacing switches it is also possible to redundantly connect the new device using the coordinates of the old one and gradually disconnect the device being replaced, since greedy routing is permissive of such addressing singularity. This process yields a "seamless" switch replacement which remains unnoticed by the servers in the topology.

A simple example for performance upgrade is depicted in Figure 8(b) where a 4-port switch at location 1 is changed to an 8-port switch. The new switch, in addition to maintaining the tessellation links, can be connected to any free switch ports




(a) (b)

Fig. 8: Figure (a) shows the process of adding 1 server to the topology. Server node 2 is shifted to an outer coordinate to make room for switch node 1, thus adding 2 new free ports to the system. One port will be used by new server node 3, and one port is left open for future use. To preserve greedy routability, we connect server 5 with server 2. Figure (b) shows how throughput capacity can be enhanced by changing a 4-port switch to a 8-port switch in the core by using a free port of switch 3.

(i.e. switch 3) or new server ports in the topology (i.e. server 4). The 4-port switch can be reused at an outer layer of the topology.

Similarly to [9], [15], *Poincaré* leverages diverse bandwidth capabilities of current router devices, namely by connecting hosts through 1 Gbps ports to ToR switches and using higher capacity 10 Gbps links in the inner layers of the topology. Using such building blocks allows designers to easily adjust oversubscription to the desired level.

We further elaborate on the simple performance enhancing algorithm shown in Section II-B by using different connection radius $r_{sw} = 4.3$ and $r_{se} = 3.3$ in case of switches and servers respectively, except if the density of the topology is indicated. We can maximize server port counts to reduce costs depending on the tessellation (i.e. in a 4-decagon tessellation allowing server port count to be 4 or less). We note that various finer-grained optimization methods (in terms of throughput, cost, etc.) would further enhance *Poincaré* overall performance, however such techniques are not in the scope of this paper.

## V. Performance Evaluation

After getting through the detailed description of the basic mechanisms in *Poincaré*, we now turn to analyze its performance according to diverse metrics via simulation. First, we describe the simulation environment and our general traffic scenario. Next, the results of multipath and multicast performance simulations are provided and we analyze the inherent fault tolerance of the architecture.

### A. Throughput

To evaluate *Poincaré*'s throughput, we use a flow-level traffic simulator implementing fat-tree and *Poincaré* routing. We compare the aggregate throughput of the two systems as the total shuffled data divided by the completion time of flows.

All topologies contain 4000 servers with varying number of switches and the results are averaged over 10 simulation runs. We adapt a permutation traffic matrix from [26] where every host sends 10 MB data to another host, and no host receives more than one flow. Table IV shows the results of the traffic simulation. It can be seen that there is a trade-off between the number of switch ports and the incurred aggregate throughput. Furthermore the performance of the system can be enhanced by high capacity links in the core. The table shows three cases for *Poincaré*; the first is a budget-friendly topology with few switch ports and few high capacity links. The second row shows a configuration equivalent to a similarly sized fat-tree DC in terms of throughput performance. Finally, we show an enhanced configuration with a higher switch port count and slightly more high capacity links. We also indicate the throughput results for different fat-tree configurations. Although the latter two configurations represent a notably different cost (see Sec. VII), they perform similarly to each other.

**Traffic distribution.** In Figure 9 we show the distribution of traffic flowing through links. The figures show that if the topology is more dense, then traffic in the middle is distributed evenly across all core links.

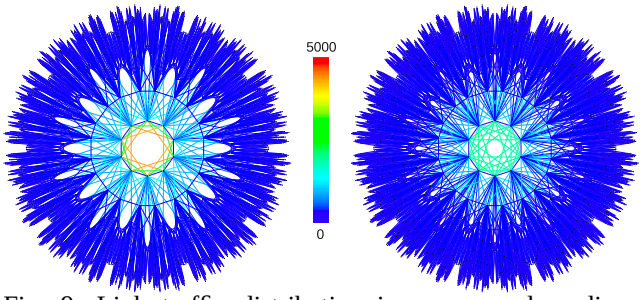



Fig. 9: Link traffic distribution in sparse and medium 4-decagon topologies.

**Forwarding burden of servers.** In *Poincaré* server nodes are also actively participating in routing to ensure greedy connectedness throughout the system. Table V demonstrates how many external flows need to be routed through the end hosts in a all-to-all communication pattern. Contrarily to the server-based routing in Bcube, *Poincaré* imposes only a small forwarding burden which can be easily managed by hosts without additional resources.

| fat-tree | *Poincaré* (4-decagon) | Bcube |
|---|---|---|
| 0 | 1386.42 | 13299 |

TABLE V: Average forwarding burden on servers for the all-to-all communication pattern on a 4096 server topology.

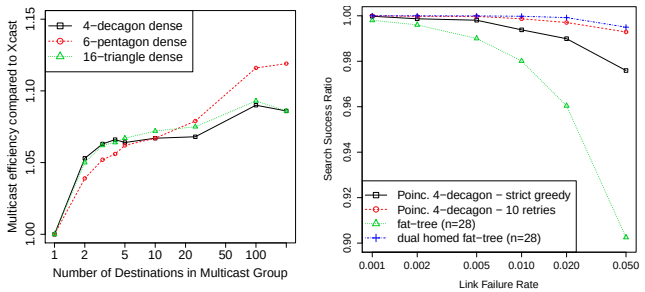
### B. Load-balancing and multicast performance

**Load-balancing.** The simple greedy multipath extension algorithm is not always able to generate edge disjoint paths on-demand. This can be regarded as a trade-off for the minimal overhead requirement of the routing algorithm (and budget friendliness). We demonstrate that the algorithm proves to be a practical and efficient load balancing approach. Table VI shows the throughput simulation results of the multipath routing algorithm. We choose random pairs of servers in 4000-server topologies, and send 100 different flows of 1MB data from the same source to the same destination. Results are averaged over 1000 different source-destination pairs. In contrast to fat-tree, where the bottleneck is the access link speed of the server, *Poincaré* leverages the multiple disjoint paths between the source and destination servers. We note that *Poincaré* routing could be also augmented with a multipath congestion control algorithm for further improvement. Smart flow management issues constitute important future work for us.

| Topology | Avg. Multi. Throughput | Var | Min. | Max. |
|---|---|---|---|---|
| *Poincaré* fat-tree equiv. | 1527.41 | 406.43 | 1000 | 2898.55 |
| fat-tree ($n = 32$)) | 1000 | 0 | 1000 | 1000 |

TABLE VI: Comparison of multipath throughput capability of 4000 server fat-tree and *Poincaré* systems (Mbps).

**Multicast performance.** The efficiency of the greedy multicast algorithm is compared to Xcast in Figure 10a. The plot shows the ratio of overall link loads when routing flows to the same set of destinations in case of Xcast and greedy multicast working modes. A factor of 1.1 means that 10% less link capacity is used by greedy multicast when sending to the same destinations.




(a) Multicast algorithm efficiency compared to Xcast.

(b) Greedy search and fat-tree search success ratio versus random link failure rates.

Fig. 10

### C. Effects of link failures on greedy search

Here we show how *Poincaré* copes with random link failures. Figure 10b shows that the resilience of the architecture is remarkable in case of realistic link failure rates [15]. We simulate random link failure events in the topology and measured the overall success ratio of greedy routing for 50000 source-destination pairs. The plot also shows the failure handling feature of greedy search, retrying to find a path for a maximum of 10 times, which improves routing success. To compare the results to fat-tree based topologies, we also plot the probability of host pairs remaining connected in these structures at the presence of link failures which is an upper bound on the success rate.

## VI. Prototype Implementation and Measurements

In order to demonstrate and further evaluate *Poincaré*, we have implemented a prototype in OpenFlow and carried out

| Topology | Switches | Server ports | Switch ports | High capacity links | Aggregate throughput | Avg. per server throughput | Runtime (ms) |
|---|---|---|---|---|---|---|---|
| *Poincaré* economic | 640 | 13510 | 10448 | 1200 | 239913.37 | 143.78 | 1334.67 |
| *Poincaré* fat-tree equiv. | 640 | 13510 | 14598 | 2350 | 494473.45 | 280.56 | 649.33 |
| *Poincaré* high throughput | 640 | 13510 | 15878 | 3500 | 558779.28 | 364.32 | 573.17 |
| fat-tree ($n = 28$)) | 980 | 4000 | 25952 | 0 | 471914.01 | 265.71 | 680.5 |
| fat-tree ($n = 32$)) | 1280 | 4000 | 36768 | 0 | 473971.33 | 265.36 | 680.17 |

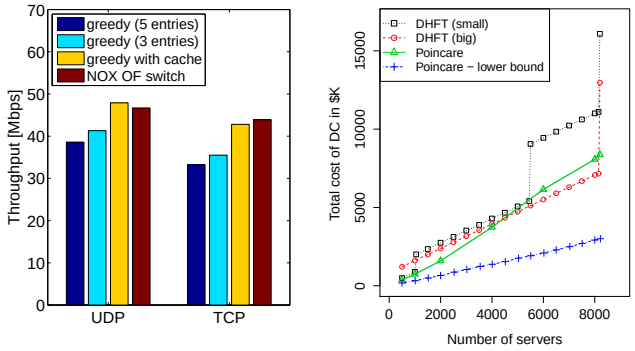
TABLE IV: Throughput (Mbps) comparison of 4000-server fat-tree and *Poincaré* DCs with different topology parameters.

performance measurements.

The OpenFlow specification [29] aims at enabling network innovation by strictly separating forwarding and control logic. Forwarding mechanisms including vendor-specific, proprietary solutions are dedicated to network devices which can be controlled through an open interface by separate controller entities. Network devices (OpenFlow switches) forward packets based on matching certain fields of packet headers with flow table entries while flow tables are maintained by controllers. In this framework, we have implemented greedy routing as a novel forwarding mechanism added to the OpenFlow reference switch v1.0 [30] working as follows. The flow table of a switch consists of one entry for all ports storing the coordinates of the corresponding neighbor and an additional one describing its own position. We use the destination MAC field in the packet header for storing coordinates and the forwarding mechanism takes this single field into account. As the OpenFlow reference switch requires, we store greedy rules in the linear flow table (wildcarded rules). Instead of standard matching, greedy forwarding calculates the distance between destination node (coordinates from incoming packet) and all neighbors (coordinates from flow entries), and finally, the entry with the minimum distance is chosen (match) and its action is executed. This means that the packet is forwarded to the closest neighbor or dropped when no closer node can be found. We emphasize that this forwarding mechanism results in a *fixed size flow table* bounded by the port number plus one. In addition, we use NOX v0.9.0 [31] as OpenFlow controller which plays a role only in the bootstrap phase; a special greedy application has been implemented in order to add flow entries to switches based on topology information.

The basic greedy forwarding mechanism can be enhanced by caching active flows in a hash table. This improvement yields faster forwarding (after the first packet, the standard exact matching is applied) at the cost of increased flow tables and memory usage. Both greedy forwarding methods, basic and enhanced, have been ported into OpenWRT [32] firmware (trunk version, bleeding edge, r26936) operating on TP-Link TL-WR841ND commodity switches. [6] In order to handle link failures, a lightweight greedy daemon has been implemented detecting link up/down events. [7] In case of link failure, the corresponding greedy flow entry is deleted by the `dpctl` tool through the local control interface of the switch, while link up event causes the restoration of the entry.

Our testbed environment consisted of the aforementioned TP-Link switches with 4+1 ports, PCs (Intel Core i3-530 CPUs at 2.93GHz, 2GB of RAM, running Debian GNU/Linux Squeeze with kernel 2.6.32-5) and a NOX v0.9.0 OpenFlow controller operating in out-of-band control mode on a separate management network.




(a) Performance on a single switch (b) Cost comparison of fat-tree and *Poincaré* while adding servers incrementally

Fig. 11

Figure 11a shows the results of simple performance measurements between two hosts connected to a single TP-Link device running different versions of OpenFlow switch. Throughput has been measured by `iperf` for UDP and TCP (Cubic) traffic, as well. On the one hand, performance of the enhanced version of greedy forwarding (with caching) is very similar to the standard switch application implemented in NOX. Only the first packet suffers from increased delay in both cases. For the standard OpenFlow switch, the controller-to-switch communication (packet in, flow mod) causes the delay, while greedy forwarding induces additional operations for the first packet. On the other hand, the basic greedy algorithm, calculating distances for all packets, shows 10-20% performance degradation depending on the number of flow entries. More specifically, the throughput of a switch with 3 and 5 entries are plotted, respectively. It should be emphasized that the switch is able to operate with fixed size flow tables at the cost of only a moderate performance degradation, even in case of our low-end COTS device with very limited computational power.

Furthermore, we have built (a relevant subset of) a *Poincaré* topology. The testbed topology consisting of four switches and two hosts (and a NOX controller) is shown in Figure 6. In this network environment, the link failure recovery mechanism of *Poincaré* is demonstrated. PC2 generates UDP traffic to PC1 at a constant bit rate and the incoming packets with timestamps are logged at the receiver and plotted in Figure 12. At 7.1 sec the indicated link on the original greedy path is unplugged

[6]This TP-Link model requires some modification in the firmware, more exactly the MAC learning function has to be disabled in the kernel driver of the switch (ag71xx_ar7240.c), and an extension is also necessary for correct port status detection.

[7]Current version of OpenFlow uses LLDP for link state detection which is implemented in the controller. Therefore, this approach can be used only for small networks, and according to our experiments, the minimum timeframe is around $4 - 5$ sec even in very simple testbeds.

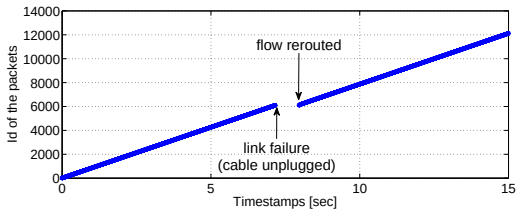


Fig. 12: Failure recovery

(link failure), thus PC1 cannot receive packets. The greedy daemons running on Switch 1 and 2 detect the link-down event and Switch 1 updates its flow table to the alternative path via Switch 4. This strictly local failure recovery mechanism achieves on-demand flow rerouting within 1 sec *with zero communication overhead*: flow restoration only depends on the speed of link event detection.

## VII. Cost Considerations: Cabling, Initial Rollout and Expansion

### A. Cabling

A practical and economical requirement of data center designs is to exhibit reasonable cabling complexity. *Poincaré* achieves this by implementing the locality of the edges since it connects nodes that are close in the hyperbolic space. If we use an $(n, k)$ tessellation as a minimum topology, the following arrangement hints at low cabling complexity. Form $k$ rows of racks each containing one top level $t_i$ switch and the servers and switches to whom $t_i$ is the closest among the $k$ top level switches. In this case most of the cables reside in one row since it can be shown that only $O(\log n)$ edges are "cross-row" edges, and hence easier to implement [33]. Similarly racking servers and switches based on space proximity should result in a moderate cabling burden.

### B. Initial rollout and incremental expansion

As it was previously shown in Section II-B *Poincaré* is well-suited for starting small and upgrading incrementally. As for an initial build-out, a *Poincaré*-based data center can be constructed of any reasonable number of servers and switches of arbitrary port count. This flexibility enables start-ups and other budget-conscious companies to restrict their initial investment (low CAPEX). To put *Poincaré* in a cost perspective, we performed a cost comparison with the dual homed fat-tree design. Our cost model is based on [22] using regular ($100) and high capacity ($250) switch ports, and also server ports ($100) as our main cost elements. The cost of cabling is also taken into account, but we find that cabling costs are marginal in compliance with [22]. Suppose that a start-up wants to build a private DC based on a fat-tree structure, starting with 500 servers. We considered two scenarios: a) minimum entry cost and b) easily upgradable system for the forseeable future with a larger initial CAPEX. We assume that upgrades are *incremental and continuous* over time. Figure 11b shows the total cost of the company's DC while expanding from 500 to 8200 servers. It is shown that by starting with a small sized fat-tree topology ($n = 16$) the incurred entry cost is low. However, when reaching maximal tree size (1024 servers), the DC requires a structural reconfiguration of changing every switch in the system. This means that the next upgrade could cost the same as the whole system did until this point. On the other hand, a larger initial fat-tree could pose prohibitive entry costs for a small company, as high as double or triple of the smaller, but equal performance tree. Moreover, this larger tree would also reach its ceiling (note the jump after 8000 servers).

The two lines corresponding to *Poincaré* show a) the minimum amount of money needed to build a working system and more importantly b) a system that is equivalent to a fat-tree system in terms of throughput. In the first case the initial DC has a very low entry cost, and it can be smoothly upgraded in small increments. In the second case, the "iso-throughput" *Poincaré* DC can be built with lower entry cost and stays cheaper than or at worst comparable to its fat-tree counterpart.

## VIII. Related Work

Recently hyperbolic geometry is gaining more focus when it comes to explaining the intricate structure of real world complex networks [19] [34] and designing networks with extreme navigational properties [35] [36] [28]. Moreover, the hyperbolic embedding of the AS-level Internet topology has been computed to illustrate the effectiveness of the hyperbolic space in real routing situations [37]. However, current proposals for generating networks embedded in the hyperbolic space cannot be directly used in data centers since they cannot guarantee e.g., $100\%$ greedy packet delivery, symmetric topology, plug $\&$ play configuration, incremental upgradability and constant upper bounded node degree with logarithmic diameter.

Several data center architectures have been proposed recently. Data centers based on fat-tree topologies [18], [15], [14] are built using commodity switches arranged in three layers, namely core, medium (aggregation), and server (edge) layers. The structure is also known as Clos topology. Portland [14] is a scalable, fault tolerant, but complex layer-2 data center fabric built on multi-rooted tree topologies which supports easy migration of virtual machines. VL2 [15] is an agile data center fabric based on end-host routing with properties like performance isolation between services, load balancing, and flat addressing. The topology of BCube [7] is generated using a recursive algorithm. BCube is intended to be used in container based, modular data centers, with a few thousand servers. MDCube proposes a method to interconnect these containers to create a mega-data center [8]. An initial design of an asymmetric, scale-free network inspired data center structure was proposed in [38].

The above designs are not geared towards incremental upgradability. The LEGUP proposal tackles the problem of expanding some existing data centers, adding heterogeneity to Clos networks [9]; however, its performance for small upgrades is unknown, as is the impact of the added heterogeneity on routing performance. Jellyfish suggests to use random networks as underlying topology. Due to the random structure, incremental expandability of a Jellyfish DC is adequate; however, routing is identified as an open research challenge

[10]. A similar argument holds in the case of REWIRE [11] as well.

## IX. Conclusion

In this paper we introduced *Poincaré*, a data center architecture embedded into the hyperbolic plane. *Poincaré* uses greedy routing to forward packets between any pair of servers and due to its topological properties always routes along optimal paths. Moreover *Poincaré*'s routing relies only on local decisions and does not require complicated routing tables to be stored and sweaty routing protocols to be run. We showed that in the hyperbolic space greedy routing effectively exploit network redundancy which bards *Poincaré* with excellent failure tolerance and recovery. Our extensions of greedy routing provides multipath and multicast possibilities hereby boosting one-to-one and one-to-many communication performance. Satisfying our basic requirement *Poincaré*'s structure and performance is easy to upgrade and can be done with arbitrary granularity, servers can be added one-by-one and every single link improves performance without additional configuration. Our feasibility analysis indicates that *Poincaré* provides comparable throughput in comparison with with fat-tree topologies and its flexibility enables to start from a very cheap but working DC and incrementally upgrade to any desired performance level, hereby significantly lowering entering costs.

## References


[1] F. Research, "Market overview: Private cloud solutions, q2 2011," *http://www.forrester.com/rb/Research/market_overview_private_cloud_solutions2C_q2_2011/q/id/58924/t/2*.

[2] 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*.

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

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

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

[6] ——, "Facebook server count: 60,000 or more," http://goo.gl/79J4, Jun. 2010.

[7] 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," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[8] 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*. New York, NY, USA: ACM, 2009, pp. 25–36.

[9] 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*. New York, NY, USA: ACM, 2010, pp. 14:1–14:12. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921187

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

[11] A. Curtis, T. Carpenter, M. Elsheikh, A. Lopez-Ortiz, and S. Keshav, "Rewire: An optimization-based framework for data center network design," Technical Report CS-2011-21, University of Waterloo, Tech. Rep., 2011.

[12] R. Perlman, D. Eastlake 3rd, D. Dutt, S. Gai, and A. Ghanwani, "Routing Bridges (RBridges): Base Protocol Specification," RFC 6325, Internet Engineering Task Force, Jul. 2011.

[13] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: a scalable ethernet architecture for large enterprises," in *ACM SIGCOMM 2008*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 3–14.

[14] 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*. New York, NY, USA: ACM, 2009, pp. 39–50.

[15] 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*. New York, NY, USA: ACM, 2009, pp. 51–62.

[16] DataCenterDynamics, "Increasing infrastructure utilization rates is key to cloud providers' success," *http://goo.gl/ey334*, 2010.

[17] P. Fraigniaud and C. Gavoille, "Routing in trees," in *ICALP '01*, 2001, pp. 757–772.

[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM '08*. New York, NY, USA: ACM, 2008, pp. 63–74.

[19] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguñá, "Hyperbolic geometry of complex networks," *Physical Review E*, vol. 82, no. 3, p. 036106, 2010.

[20] J. Anderson, *Hyperbolic geometry*. Springer Verlag, 2005.

[21] David E. Joyce, "Hyperbolic Tessellations: http://aleph0.clarku.edu/~djoyce/poincare/PoincareApplet.html."

[22] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, "A cost comparison of datacenter network architectures," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 16.

[23] O. Paridaens, Y. Imai, R. Boivie, W. Livens, and D. Ooms, "Explicit multicast (xcast) concepts and options," *RFC 5058 IETF*, 2007.

[24] A. Aloisio, V. Izzo, and S. Rampone, "Fpga implementation of a greedy algorithm for set covering," in *Real Time Conference, 2005. 14th IEEE-NPSS*. IEEE, 2005, pp. 5–pp.

[25] P. Elias, A. Feinstein, and C. Shannon, "A note on the maximum flow through a network," *Information Theory, IRE Transactions on*, vol. 2, no. 4, pp. 117–119, 1956.

[26] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM '11*, 2011.

[27] H. Frey and I. Stojmenovic, "On delivery guarantees of face and combined greedy-face routing in ad hoc and sensor networks," in *Proceedings of the 12th annual international conference on Mobile computing and networking*. ACM, 2006, pp. 390–401.

[28] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 1647–1655.

[29] 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.*, vol. 38, no. 2, pp. 69–74, 2008.

[30] OpenFlow Switch, "http://openflow.org/."

[31] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[32] OpenWrt, "http://openwrt.org/."

[33] J. Mudigonda, P. Yalagandula, and J. C. Mogul, "Taming the flying cable monster: a topology design and optimization framework for data-center networks," in *Proceedings of USENIX ATC'11*. USENIX, 2011.

[34] M. Boguna, D. Krioukov, and K. C. Claffy, "Navigability of complex networks," *Nat Phys*, vol. 5, no. 1, pp. 74–80, 2009. [Online]. Available: http://dx.doi.org/10.1038/nphys1130

[35] R. Kleinberg, "Geographic routing using hyperbolic space," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. Ieee, 2007, pp. 1902–1909.

[36] F. Papadopoulos, D. Krioukov, M. Bogua, and A. Vahdat, "Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, pp. 1–9.

[37] M. Boguna, F. Papadopoulos, and D. Krioukov, "Sustaining the internet with hyperbolic mapping," *Nature Communications*, vol. 1, no. 6, pp. 1–8, 2010.

[38] L. Gyarmati and T. A. Trinh, "Scafida: a scale-free network inspired data center architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 4–12. [Online]. Available: http://doi.acm.org/10.1145/1880153.1880155