Routing Algorithms for Recursively-Defined Data Centre Networks

Alejandro Erickson and Iain A. Stewart
School of Engineering and Computing Sciences
Durham University, Science Labs, South Road
Durham DH1 3LE, U.K.
Email: {alejandro.erickson,i.a.stewart}@durham.ac.uk

Abbas Eslami Kiasari and Javier Navaridas
School of Computer Science
University of Manchester, Oxford Road
Manchester M13 9PL, U.K.
Email: {abbas.kiasari,javier.navaridas}@manchester.ac.uk

***Abstract*—The server-centric data centre network architecture can accommodate a wide variety of network topologies. Newly proposed topologies in this arena often require several rounds of analysis and experimentation in order that they might achieve their full potential as data centre networks. We propose a family of novel routing algorithms on two well-known data centre networks of this type, (Generalized) DCell and FiConn, using techniques that can be applied more generally to the class of networks we call completely connected recursively-defined networks. In doing so, we develop a classification of all possible routes from server-node to server-node on these networks, called general routes of order t, and find that for certain topologies of interest, our routing algorithms efficiently produce paths that are up to 16% shorter than the best previously known algorithms, and are comparable to shortest paths. In addition to finding shorter paths, we show evidence that our algorithms also have good load-balancing properties.**

I. Introduction

The explosive growth of online services powered by data centres (web search, cloud computing, etc.) has motivated intense research into data centre network (DCN) design over the past decade and brought about major breakthroughs. For example, fat-tree DCNs, introduced in [1], use commodity off-the-shelf (COTS) servers and switches in a fat-tree (topology), and have resulted in an evolutionary shift in production data centres towards leaf-spine topologies, built from COTS hardware. COTS fat-tree DCNs are not a panacea, however; for example, fat-trees are difficult to scale.

Research on DCN architecture is ongoing and each new architecture invites the use of certain classes of topologies. Indirect networks, where servers are the terminals connected to a switching fabric, are the prevailing example. Fat-trees are among the topologies that can be implemented in indirect network architectures. A host of alternative topologies can be implemented as indirect networks, including random regular graphs ([2]) and butterfly networks ([3]). Likewise, the optical-switch hybrid DCN Helios ([4]) can be seen as an architecture with the capacity to accommodate a variety of topologies (both in the wired links as well as in the optical switch itself). Each architecture sets constraints on the topology in a variety of ways; for example, by the separation of switching nodes from server nodes or the number of ports in the available hardware.

The server-centric DCN (SCDCN) architecture, introduced in [5], accommodates a great variety of network topologies and has resulted in a number of new DCN designs, both derived from existing and well-understood topologies in interconnection networks as well as topologies geared explicitly towards DCNs (e.g., [5]–[10]).

Only dumb crossbar-like switches are used in an SCDCN and the servers are responsible for routing packets through the network. Therefore, the switches have no knowledge of the network topology and are only connected to servers. Servers, on the other hand, may be connected to both switches and servers. These parameters, which make up part of the SCDCN architecture, invite sophisticated topologies from abstractions as graphs, along with accompanying analyses. We are concerned primarily with routing algorithms for two well-known SCDCNs, DCell ([5]) and FiConn ([6]), and the topologies called Generalized DCell ([11,12]).

We characterise (Generalized) DCell and FiConn as a special case of *completely connected recursively-defined networks* (CCRDN), which we use to develop a classification (which, to our knowledge, is novel) of all possible routes from server-node to server-node in the DCNs (Generalized) DCell and FiConn. Our main result pertains to a specific family of routing algorithms, called `PR` (or `ProxyRoute`), which we develop with the primary aim of improving upon the originally proposed (and best known) routing algorithms, as regards hop-length. This goal is achieved with improvements as high as 16% for certain topologies and paths that are comparable, in length, to shortest paths. In addition, we give empirical evidence that the path diversity provided by `PR` does a better job of balancing load than `DCellRouting`. Hitherto, the only algorithms for balancing communication load in (Generalized) DCell and FiConn are the adaptive routing algorithms `DFR` and `TAR` presented in [5,6], so `PR` is also novel in this respect.

Two of our instances of `PR` called `GP_I` and `GP_0`, exploit the topological structure of (Generalized) DCell and FiConn in order to find short paths efficiently by means of an intelligent search (see Section V-A) of sub-structures called “proxies”. We then empirically compare the results of our intelligent versions of `PR` with a shortest path algorithm, a brute force version of `PR` and the routing algorithms that were originally proposed in [5,6,12].

We give definitions in Sections II–III, where we abstract the DCNs (Generalized) DCell and FiConn as graphs which can be characterised as CCRDNs. Section IV describes previously known routing algorithms for these DCNs, in the context of CCRDNs, and our classification of routes in CCRDNs is given in Section IV-B, as *general routes of order* t. We present our main contribution in Section V: the design of PR. Our empirical work is described and evaluated in Section VI and future avenues for research are identified in the conclusion.

II. Server-centric DCNs

Our results and experiments are concentrated on graph theoretical abstractions of certain SCDCNs. Therefore, it is appropriate that we define this abstraction precisely.

An SCDCN consists of switches, which act only as cross-bars and have no routing intelligence, and servers. These components are linked together, with the only restriction being that a switch cannot be linked directly to another switch; we assume all links are bidirectional. As such, an SCDCN is abstracted here by an undirected graph $G = (W \cup S, E)$, with two types of nodes called *switch-nodes*, W, and *server-nodes*, S. Naturally, each switch of the SCDCN corresponds to a switch-node, $w \in W$, and each server corresponds to a server-node, $x \in S$. Each link of the SCDCN corresponds to an edge e of E, which, for convenience, we shall also call a link. The condition that switch-to-switch links are not allowed implies that $\nexists (u, v) \in E$ such that $u, v \in W$. See [13] for undefined graph-theoretic terms.

Also relevant to our discussion of routing algorithms in SCDCNs is the fact that(1) packets are sent and received only by servers, and (2) packets endure a negligible amount of processing time in each switch, compared to the time spent in each server. The reason for (2) is that we assume the packet is routed in the server's operating system, either via a table look-up or computation. This could be done, e.g., by a dedicated virtual machine or a specialised hypervisor with the capability to route packets. In any case, we may assume that with today's COTS servers, a packet spends much more time at servers than in switches.

The outcome of (1) is that we need only discuss routing algorithms that construct paths whose endpoints are server-nodes. That is, a *route* on G is a path whose endpoints are server-nodes. The outcome of (2) is that a *hop* from server-node to server-node is indistinguishable from one that also passes through a switch-node.

III. Recursively-Defined Networks

Our results are concerned with network topologies of a certain form that have arisen frequently in the area of inter-connection networks, and recently as SCDCNs.

Definition III.1. *A family* $\mathcal{X} = \{X(h) : h = 0, 1, \ldots\}$ *of interconnection networks is* recursively-defined *if* $X(h)$*, where* $h > 0$*, is the disjoint union of copies of* $X(h-1)$ *with the addition of extra links joining nodes in the different copies. We call a member of* $\mathcal{X}$ *a* recursively-defined network (RDN). *A family of RDNs* $\mathcal{X}$ *is a* completely-connected RDN (CCRDN) *(see, e.g., [14]) if there is at least one link joining every copy of* $X(h-1)$ *within* $X(h)$ *to every other copy.*

A. The DCNs DCell

The DCNs DCell ([5]) were the first family of SCDCNs to be proposed, and their graphs form the family of CCRDNs described below.

Fix some $n > 2$. The graph $\text{DCell}_{0,n}$ consists of one switch-node connected to n server-nodes. For $k \geqslant 0$, let t_k be the number of server-nodes in $\text{DCell}_{k,n}$. For $k > 0$, the graph $\text{DCell}_{k,n}$ consists of $t_{k-1} + 1$ disjoint copies of $\text{DCell}_{k-1,n}$, labelled D^i_{k-1}, for $0 \leqslant i \leqslant t_{k-1}$. Each pair of distinct $\text{DCell}_{k-1,n}$s is joined by exactly one link, called a *level-k link*, whose exact definition is given below, in terms of the labels of the server-nodes.

Label a server-node of a $\text{DCell}_{n,k}$, for some $k > 0$, by $x = x_k x_{k-1} \cdots x_0$, where $x_{k-1}x_{k-2} \cdots x_0$ is the label of a server-node in $D^{x_k}_{k-1}$, and $0 \leqslant x_0 < n$ and $0 \leqslant x_i < g_k$ for $i > 0$, where $g_k = t_{k-1} + 1$. The labels of $\text{DCell}_{n,k}$ are mapped bijectively to the set $\{0, 1, \ldots, t_k - 1\}$ by $uid_k(x) = x_k t_{k-1} + x_{k-1} t_{k-2} + \cdots + x_1 t_0 + x_0$. Label and uid are combined in the notation $[x_k, uid_{k-1}(x_{k-1}x_{k-2} \cdots x_0)]$.

Let $0 \leqslant x_k < y_k < t_{k-1} + 1$ be the indices of the $\text{DCell}_{k-1,n}$s labelled $D^{x_k}_{k-1}$ and $D^{y_k}_{k-1}$. A level-k link connects node $y_k - 1$ in $D^{x_k}_{k-1}$ to node x_k in $D^{y_k}_{k-1}$. This is the link $(y_k - 1 + x_k t_{k-1}, x_k + y_k t_{k-1})$.

1) Generalized DCell: The definition of the DCNs DCell generalises readily; see [11,12]. The key observation is that the level-k links are a perfect matching of the server nodes in the disjoint copies of the $\text{DCell}_{k-1,n}$s, where every pair of distinct $\text{DCell}_{k-1,n}$s is connected by a link. Many such matchings are possible. A given matching ρ_k which satisfies the stated properties defines the level-k links and is called a *ρ_k-connection rule* ([12]).

A *Generalized* $\text{DCell}_{k,n}$ inherits the definition of $\text{DCell}_{k,n}$, for $k \geqslant 0$, except that the level-k links may satisfy an arbitrary ρ_k-connection rule. Note that we insist that there be only one connection rule for each level k, so that a given family of Generalized DCells can be specified by a set of connection rules $\{\rho_1, \rho_2, \rho_3, \ldots\}$.

This is in accordance with Definition 1 in [12], with two exceptions. We model Generalized $\text{DCell}_{0,n}$ as a switch-node connected to n server-nodes, rather than modelling it as K_n, and we require $n > 2$.

In order to demonstrate the impact of different connection rules on the routing algorithms presented in Section IV, it suffices to consider just one connection rule besides the one for DCell. For this purpose, we use β-DCell, defined by the β-connection rule given in [12].

The β-connection rule is (perhaps not obviously) as follows: Let $0 \leqslant x_k < y_k < t_{k-1} + 1$ be the indices of the β-$\text{DCell}_{k-1,n}$s labelled $B^{x_k}_{k-1}$ and $B^{y_k}_{k-1}$. A level-k link connects node $y_k - x_k - 1$ in $B^{x_k}_{k-1}$ to node $t_{k-1} - y_k + x_k$ in $B^{y_k}_{k-1}$. This is the link $(y_k - x_k - 1 + x_k t_{k-1}, t_{k-1} - y_k + x_k + y_k t_{k-1})$.

B. The DCNs FiConn

One of the issues with (Generalized) $\text{DCell}_{k,n}$ is that each server-node has degree $k + 1$. This requires that each server has $k+1$ NIC ports, which is not typically the case for COTS servers when $k > 1$.

FiConn, proposed in [6], is a CCRDN that requires at most two ports per server; it uses only half of the *available* server-nodes (those of degree one) in each copy of FiConn$_{k-1,n}$ when building FiConn$_{k,n}$. This, in turn, leaves server-nodes of degree one available to build the next level. We describe FiConn below.

Fix some even $n > 3$. FiConn$_{0,n}$ is the network consisting of one switch-node connected to n server-nodes. Let b be the number of available server-nodes in FiConn$_{k-1,n}$ for $k > 0$. Build FiConn$_{k,n}$ from $b/2+1$ copies of FiConn$_{k-1,n}$, labelled F^i_{k-1}, for $0 \leqslant i \leqslant b/2$. From [6] we have that $b/2+1 = t_{k-1}/2^k + 1$, so that the label of a server-node x of a FiConn$_{k,n}$ is, expressed as the $(k+1)$-tuple $x = x_k x_{k-1} \cdots x_0$, where $x_{k-1}x_{k-2}\cdots x_0$ is a server-node in $F^{x_k}_{k-1}$ and we have $0 \leqslant x_0 < n$, but $0 \leqslant x_i < g_k$, where $g_k = b/2+1 = t_{k-1}/2^k + 1$ (diverging slightly from the labels in DCell). We have $uid_k(x) = x_k t_{k-1} + x_{k-1}t_{k-2} + \cdots + x_1 t_0 + x_0$ and $[x_k, uid_{k-1}(x_{k-1}x_{k-2}\cdots x_0)]$ to label server-nodes, once more.

Let $0 \leqslant x_k < y_k < t_{k-1}/2^k + 1$ be the indices of the FiConn$_{k-1,n}$s $F^{x_k}_{k-1}$ and $F^{y_k}_{k-1}$. A level-k link connects server-node $(y_k-1)2^k + 2^{k-1} + 1$ in $D^{x_k}_{k-1}$ to server-node $x_k 2^k + 2^{k-1} + 1$ in $D^{y_k}_{k-1}$. This is the link $((y_k-1)2^k + 2^{k-1} + 1 + x_k t_{k-1}, x_k 2^k + 2^{k-1} + 1 + y_k t_{k-1})$.

IV. Routing

CCRDNs feature a class of routing algorithms that emerges naturally from their definition, called *dimensional routing*.

A. Dimensional routing

Definition IV.1. *Let $\mathcal{X} = \{X(h) : h = 0,1,\ldots\}$ be a family of CCRDNs, and let X_h be a copy of $X(h)$, for some fixed $h > 0$. Let X^a_{h-1} and X^b_{h-1} be disjoint copies of $X(h-1)$ in X_h, and let src and dst be nodes of X^a_{h-1} and X^b_{h-1}, respectively. Since X_h is completely connected, there is a level-h link in X_h incident with a node dst' in X^a_{h-1} and a node src' in X^b_{h-1}. If $h-1=0$ then either $src = dst'$ or (src, dst') is a link, and otherwise a path P_a from src to dst' can be recursively computed in X^a_{h-1}. This same method provides a path P_b from src' to dst in X^b_{h-1}. A* dimensional routing algorithm *on $\mathcal{X}$ is one which computes paths of the form $P_a + (dst', src') + P_b$, between any source-destination pair of nodes in a member of $\mathcal{X}$, and is denoted $\mathtt{DR}_{\mathcal{X}}$. A* dimensional route *is one that can be computed by a dimensional routing algorithm.*

Remarkably (and, perhaps, unfortunately), there are topologies and source-destination pairs for which no dimensional routing algorithm computes a shortest path; a notable example is the family of WK-recursive networks ([15]), for which a shortest path algorithm is developed in [16].

1) Dimensional routing in (Generalized) DCell and FiConn: (Generalized) DCell and FiConn are CCRDNs in which each pair of disjoint copies of DCell$_{k-1,n}$ within DCell$_{k,n}$ is joined by exactly one edge. As such, there is only one choice for the edge (dst', src'), which is computed by the connection rule for level-h links. Therefore, the connection rules in Sections III-A–III-B suffice to describe dimensional routing for these DCNs.

The dimensional routing algorithms for each of these networks serves as a basis for fault-tolerant and load-balancing routing algorithms `DFR` in [5], and `TAR` in [6], and it is precisely the algorithm called Generalized `DCellRouting`, given in [12]. The former two are fault and congestion-tolerant routing algorithms that compute significantly longer paths, on average, than the dimensional routing algorithms.

B. Proxy Routing

A *general routing algorithm* on a family $\mathcal{X} = \{X(h) : h = 0,1,\ldots\}$ of CCRDNs is of the following form. Let X_h be a copy of $X(h)$, for some fixed $h > 0$. Let $X^{c_0}_{h-1}$ and $X^{c_{t-1}}_{h-1}$ be disjoint copies of $X(h-1)$ in X_h, with src_{c_0} and $dst_{c_{t-1}}$ nodes of $X^{c_0}_{h-1}$ and $X^{c_{t-1}}_{h-1}$, respectively. Let $X^{c_0}_{h-1}, X^{c_1}_{h-1}, \ldots, X^{c_{t-1}}_{h-1}$ be a sequence of copies of $X(h-1)$, where: $c_0 = a$; $c_{t-1} = b$; $c_i \neq c_{i+1}$, for $0 \leqslant i < t$; and $X^{c_i}_{h-1}$ is disjoint from $X^{c_j}_{h-1}$ whenever $c_i \neq c_j$. Let $(dst_{c_i}, src_{c_{i+1}})$ be a link from $X^{c_i}_{h-1}$ to $X^{c_{i+1}}_{h-1}$, and let P_i be paths in each $X^{c_i}_{h-1}$ from src_{c_i} to dst_{c_i}.

Every routing algorithm computes a path (we shall assume that there are no repeated nodes) of the form $P_0 + (dst_{c_0}, src_{c_1}) + P_1 + \ldots + (dst_{c_{t-2}}, src_{c_{t-1}}) + P_{t-1}$.

A *general route of order T* is one in which $t \leqslant T$ for each $X(h)$, with $h = 0,1,\ldots$ and $t = T$ for at least one of these. A *proxy route*, computed by a *proxy routing algorithm*, is a general route of order 3 (and a dimensional route is of order 2).

1) `DFR` for DCell and `TAR` for FiConn: While we do not provide full details here, we sketch the proxy-routing-like subroutine that is common to `DFR` ([5]) and `TAR` ([6]). Both `DFR` and `TAR` are adaptive routing algorithms which compute paths in a distributed manner, making decisions on the fly, based on information that is local to the current location of the packet being routed.

This subroutine computes a part of a proxy route to replace a sub-path of the intended route. In particular, a packet may bypass a level m link, e, from sub-structure D^a_{m-1} to D^b_{m-1} by re-routing through a proxy, D^c_{m-1}, with a, b, and c distinct. The decision to bypass is made when the packet arrives at e (or near e, as determined by a parameter in `DFR`), and upon its arrival in D^b_{m-1}, the packet is routed directly to its final destination.

The algorithms `DFR` and `TAR` produce much longer than `DR`, on average. The simulations in [5] show that `DFR`, although fault-tolerant, computes paths that are over 10% longer than the shortest paths, on average, even with as little as 2% failures. The maximum length of a route computed by the implementation of `TAR` in [6] (Theorem 7) is $2\cdot 3^k - 1$, whilst it is $2\cdot 2^k - 1$ for `DR` (called `TOR` in [6]). This is reflected in their simulations of random and burst traffic, where `TAR` computes paths that are 15-30% longer, on average, than those computed by `DR`.

V. Proxy routing in DCell and FiConn

We propose that proxy routing be used more broadly than it is in DFR and TAR, and with the primary goal of efficiently computing short paths, rather than fault-tolerance and balancing load, by applying it in a fundamentally different manner: firstly, we seek to compute a proxy route at the outset, rather than building the route piecemeal; secondly, we use this pre-planning in order to find a proxy route that offers a high degree of savings over the dimensional route.

One reason for focusing on $t \leqslant 3$ is that visiting each $X^{c_j}_{m-1}$, for $0 < j < t-1$, has an associated cost, and when m is small, as it is when our graphs represent DCNs with a realistically deployable number of servers, it becomes less likely that general routes with $t > 3$ will be useful. Furthermore, the methods of searching for a "good" proxy that we explore here may become impractical for $t > 3$, because the search space of potential (multiple) proxies is much larger.

Henceforth we use $\mathcal{G}$-Cell in place of (Generalized) DCell and FiConn whenever we make statements or arguments that apply to all of these.

The following lower bound on the hop-length of a general route of order t is obvious.

Lemma V.1. *Let src and dst be server-nodes in a $\mathcal{G}$-Cell$_{k,n}$, with $k > 0$, such that src is in D^a_{k-1} and dst is in D^b_{k-1}, with $a \neq b$. A general route of order t has length at least $2t-3$. In particular, a dimensional route has length at least 1 and a proxy route has length at least 3.*

The remainder of our paper is a comparative empirical analysis of several versions of PR, given in Algorithm 1.

Algorithm 1 PR for $\mathcal{G}$-Cell returns a proxy route if it finds one that is shorter than the corresponding dimensional route.

```
Require: $src$ and $dst$ are server-nodes in a $\mathcal{G}$-Cell$_{k,n}$.
  function PR($src, dst, m$)
    if $m > 0$ and both $src$ and $dst$ are in the same
       copy of $\mathcal{G}$-Cell$_{m-1,n}$ then
      return PR($src, dst, m-1$)
    end if
    $D^c_{m-1} \leftarrow$ GP($src, dst, m$).
    if $D^c_{m-1} = null$ then
      return DR($src, dst$).
    else
      $D^a_{m-1} \leftarrow$ the $\mathcal{G}$-Cell$_{m-1,n}$ containing $src$.
      $D^b_{m-1} \leftarrow$ the $\mathcal{G}$-Cell$_{m-1,n}$ containing $dst$.
      $(a^c, c^a) \leftarrow$ the link from $D^a_{m-1}$ to $D^c_{m-1}$.
      $(c^b, b^c) \leftarrow$ the link from $D^c_{m-1}$ to $D^b_{m-1}$.
      return
            PR($src, a^c, m-1$) + $(a^c, c^a)$+
            PR($c^a, c^b, m-1$) + $(c^b, b^c)$+          (1)
            PR($b^c, dst, m-1$).
    end if
  end function
```

A. GP: GetProxy

GP is the subroutine of PR that computes the proxy used in Expression (1), if a proxy is to be used. That is, GP returns either a proxy sub-$\mathcal{G}$-Cell, D^c_{m-1}, or it returns *null*. Obviously, the performance of PR (and its success in producing a shorter route than DR) depends on the proxy returned by GP and how GP is implemented.

Ideally GP would instantly compute a unique proxy sub-$\mathcal{G}$-Cell D^c_{m-1}, if it exists, such that the proxy route through D^c_{m-1} is the shortest one possible. Such an algorithm is unknown to us.

Our strategy, however, is widely applicable, as regards different connection rules and path diversity. Every version of GP that we explore is of the following form. Let (src, dst, m) be the inputs to GP. If $m = 0$, GP outputs *null*; otherwise, let $m > 0$, so that src is in D^a_{m-1} and dst is in D^b_{m-1}, for some a not equal to b. GP computes a set of *candidate proxies*, $\{D^{c_0}_{m-1}, D^{c_1}_{m-1}, \ldots, D^{c_{R-1}}_{m-1}\}$ (taken from the set of all potential proxy $\mathcal{G}$-Cell$_{m-1,n}$s), and then finds a c_i for which the path in Expression (1) is shortest (replacing c by c_i), by constructing the paths explicitly. If the set of candidate proxies is empty, then GP returns *null*.

The key observation is that we must minimise the number of candidate $D^{c_i}_{m-1}$s in order to reduce the search space. Our goal is to identify and evaluate general techniques towards this end, and not to catalogue all of the ways to tune GP. Some more complicated techniques are avoided because there is no room to discuss them in this paper; for example when routing in a $\mathcal{G}$-Cell$_{k,n}$ we only apply PR at the top level, whereas slightly shorter paths can be obtained, on average, by using proxy routes in the recursive calls to PR at Expression (1). Other techniques are avoided because they are evidently unprofitable; for example, a much larger search is encountered if GP computes proxy paths for each proxy candidate. We describe three strategies for generating the candidate proxies below.

1) GP_E as an exhaustive search: A proxy DCell$_{m-1,n}$ D^c_{m-1} can be obtained, naïvely, if GP is implemented as an exhaustive search; that is, we perform the steps described in Section V-A for every c in $\{0, 1, \ldots, t_{m-1}\} \backslash \{a, b\}$. Measuring the length of each proxy route has an associated cost, but GP_E provides the optimal proxy route with top-level proxies only against which to test the two strategies given below.

2) GP_I as an intelligent search: We propose a general method for reducing the proxy search space, based on the labels of src and dst. In particular, we look at proxies D^c_{k-1} whose relationship to D^a_{k-1} and D^b_{k-1} is such that at least one of the routes computed by the recursive calls to PR is confined to a $\mathcal{G}$-Cell$_{k-2,n}$ (see Fig. 1).

We first give some notation. Henceforth, let D_k be an instance of $\mathcal{G}$-Cell$_{k,n}$, and let DR be the dimensional routing algorithm on $\mathcal{G}$-Cell. For clarity of exposition we describe a method for selecting a proxy D^c_2 when routing in a $\mathcal{G}$-Cell$_{k,n}$, with $k = 3$, but the notation extends to all $k > 1$.

Let src and dst be nodes in a $\mathcal{G}$-Cell$_{3,n}$, with $src = a_3a_2a_1a_0$ and $dst = b_3b_2b_1b_0$, so that $uid_3(src) = t_2a_3 +$

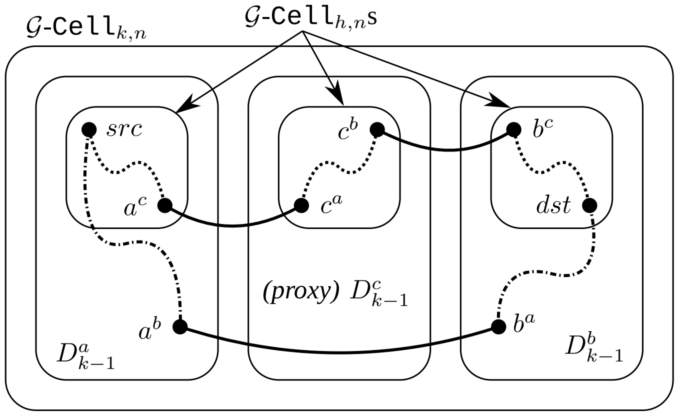

Fig. 1. Strategy for GP_I, where $h = k - 2$, and for GP_0 where $h = 0$: select c such that at least one sub-path is contained in a $\mathcal{G}$-$\mathtt{Cell}_{h,n}$. Solid arcs represent links, and dashed or dotted curves represent paths.

$t_1a_2 + t_0a_1 + a_0$ and $uid_3(dst) = t_2b_3 + t_1b_2 + t_0b_1 + b_0$. Let $a_3 \neq b_3$, and note that without loss of generality, we may assume $a_3 < b_3$.

Our convention for denoting the link between two sub-$\mathcal{G}$-$\mathtt{Cell}$s is as follows: let D^α_2 and D^β_2 be $\mathcal{G}$-$\mathtt{Cell}_{2,n}$s and recall that we may write $[\alpha, uid_2(v)]$ for a node $v = \alpha v_2v_1v_0$ in D^α_2, where $uid_2(v) = t_1v_2 + t_0v_1 + v_0$. Let $([\alpha, \alpha^\beta], [\beta, \beta^\alpha])$ be the link from D^α_2 to D^β_2, with $\alpha^\beta = \alpha^\beta_2\alpha^\beta_1\alpha^\beta_0$, and similarly for $\beta^\alpha = \beta^\alpha_2\beta^\alpha_1\beta^\alpha_0$.

GP_I builds its set of proxy candidates on the condition that the source and destination are not near to each other. Let $a = a_3$ and let $b = b_3$. GP_I outputs *null* if $[a_3, a^b]$ is a server-node of $D^{a_2}_1$ or $[b_3, b^a]$ is a server-node of $D^{b_2}_1$. That is, when $a_2 = a^b_2$ and $b_2 = b^a_2$.

Provided the above condition is avoided, we then select a proxy D^c_2 to be a candidate, when c is such that one of the three sub-paths, $\mathtt{PR}(src, [a, a^c])$ or $\mathtt{PR}([c, c^a], [c, c^b])$ or $\mathtt{PR}([b, b^c], dst)$, is short; specifically, if at least one of the three sub-paths is contained inside a single $\mathcal{G}$-$\mathtt{Cell}_{1,n}$. That is, c satisfies at least one of the following three properties (in a non-trivial way; see discussion below):

$$src \text{ and } [a, a^c] \text{ are in the same } D_1 : a_2 = a^c_2 \quad (2)$$
$$[c, c^a] \text{ and } [c, c^b] \text{ are in the same } D_1 : c^a_2 = c^b_2 \quad (3)$$
$$[b, b^c] \text{ and } dst \text{ are in the same } D_1 : b_2 = b^c_2, \quad (4)$$

where $a^c_2 = \lfloor a^c/t_1 \rfloor$ and similarly for c^a_2 and b^c_2. Clearly for any $\mathcal{G}$-$\mathtt{Cell}$ we can verify whether a proxy candidate D^c_2 satisfies one (or more) of the Properties (2)–(4), since the numerators are computed directly from the various connection rules of each $\mathcal{G}$-$\mathtt{Cell}$. However, we wish to compute the set of values c which satisfy Properties (2)–(4) in constant time.

The floor function yields that $\lfloor a^c/t_1 \rfloor = a_2$ if, and only if, $a_2t_1 \leqslant a^c < (a_2 + 1)t_1$. It happens that for our connection rules (see Sections III), a^c is piecewise linear (as a function of c), and similarly for b^c, c^a, and c^b, with exactly three cases: namely, $c_3 < a_3 < b_3$; $a_3 < c_3 < b_3$; and, $a_3 < b_3 < c_3$ (where the case $b_3 < a_3$ is treated by swapping src and dst). As a result of this, the set of values c which satisfy Properties (2)–(4) can be computed very efficiently for our connection rules as the union of, at most, a constant number of intervals (see Table I). Note that for the connection rules explored in this paper Property (3) is redundant because it does not narrow the search space; for certain pairs (a, b), all c satisfy Property (3), while no c satisfies it for other pairs.

For the case $k = 3$ and the connection rules for DCell, β-DCell, and FiConn, GP_I considers a small set with around t_1 or $2t_1$ candidate proxies. More generally, a close inspection of Properties (2) and (4) reveals that they each yield exactly t_2 (possibly disjoint) candidate proxies for Generalized DCell and at most t_1 candidate proxies for FiConn. Due to space constraints we omit a full discussion of this, but we remark that a better understanding of this aspect of proxy routes may shed light on the sophisticated relationship between the connection rule and various distance metrics on $\mathcal{G}$-$\mathtt{Cell}$.

3) GP_0 level-0 proxy search: We note that for a $\mathcal{G}$-$\mathtt{Cell}_{k,n}$, with $k = 2$, the proxy candidates D^c_1 computed by GP_I are simply those for which a^c is in the same copy of $\mathcal{G}$-$\mathtt{Cell}_{0,n}$ as src or b^c is in the same copy of $\mathcal{G}$-$\mathtt{Cell}_{0,n}$ as dst or c^a and c^b are in the same copy of $\mathcal{G}$-$\mathtt{Cell}_{0,n}$. GP_0 mimics GP_I, but computes the set of proxies that satisfy at least one of the aforementioned properties, in place of Properties (2)–(4). It is applied only to $\mathcal{G}$-$\mathtt{Cell}_{k,n}$ with $k > 2$.

4) Implementation notes: The savings in hop-length and the benefit to load-balancing come at the cost of searching proxy candidates, whose number is given by $\bar{p}$ in Fig. 3. For each proxy candidate c, the lengths of sub-paths $\mathtt{PR}(src, [a, a^c])$ or $\mathtt{PR}([c, c^a], [c, c^b])$ or $\mathtt{PR}([b, b^c], dst)$ must be computed; hence the reason for devising GP_I and GP_0 with the object of minimising c. Once GP* is "tuned" to suit a certain application and network size, however, there are several choices for how it can be implemented. How exactly this is done depends on the size of the network and the nature of the application, but we shall remind ourselves of some of the available tools.

The most naïve method is to compute the route at the source-node, by computing the candidate paths explicitly, and measuring their length, however, other methods such as table look-ups must to be considered.

GP_I, in particular, leverages the fact that $\mathcal{G}$-$\mathtt{Cell}_{k,n}$s grow double-exponentially in k in order to find proxy candidates D^c_{k-1} that are linked to the same copy of $\mathcal{G}$-$\mathtt{Cell}_{k-2,n}$ as src or dst. This has a secondary benefit; namely, $\mathcal{G}$-$\mathtt{Cell}_{k-2,n}$ (and even $\mathcal{G}$-$\mathtt{Cell}_{k-1,n}$) is small, relative to $\mathcal{G}$-$\mathtt{Cell}_{k,n}$, and this makes table look-ups feasible for storing the lengths of paths within each copy of $\mathcal{G}$-$\mathtt{Cell}_{k-2,n}$, and possibly within each copy of $\mathcal{G}$-$\mathtt{Cell}_{k-1,n}$. The whole table must be replicated at each server-node to be used this way, but this is still much smaller than storing every (src, dst)-pair. For example, there are $24,492^2 = 599,858,064$ such pairs in DCell$_{3,3}$, and $g_3t^2_2 = 157 * 156^2 = 3,820,752$ pairs confined to sub-DCell$_{2,3}$s, and $g_3g_2t^2_1 = 157 * 13 * 12^2 = 293,904$ pairs confined to sub-DCell$_{1,3}$s (see Table II).

In addition to table look-ups, we also leverage the fact that paths are computed for flows, rather than packets, and in

route $\setminus$ c	$c_3 < a_3 < b_3$	$a_3 < c_3 < b_3$	$a_3 < b_3 < c_3$
$a_3a_2a_1a_0$ to $[a, a^c]$	$\lfloor c_3/t_1 \rfloor = a_2$	$\lfloor c_3 - 1/t_1 \rfloor = a_2$	$\lfloor c_3 - 1/t_1 \rfloor = a_2$
$[c, c^a]$ to $[c, c^b]$	$\lfloor a_3 - 1/t_1 \rfloor = \lfloor b_3 - 1/t_1 \rfloor$	$\lfloor a_3/t_1 \rfloor = \lfloor b_3 - 1/t_1 \rfloor$	$\lfloor a_3/t_1 \rfloor = \lfloor b_3/t_1 \rfloor$
$[b, b^c]$ to $b_3b_2b_1b_0$	$\lfloor c_3/t_1 \rfloor = b_2$	$\lfloor c_3/t_1 \rfloor = b_2$	$\lfloor c_3 - 1/t_1 \rfloor = b_2$

TABLE I
PROPERTIES (2)–(4) APPLIED TO DCELL$_{3,n}$.

certain applications may be re-used for multiple flows among a set of server-nodes that is small, relative to the entire network. In addition, each time we compute a proxy path, we may identify multiple viable proxies (the context of the application and network size defines what this means), and hence, path diversity comes at no extra cost. We may choose from several paths at random, send a probe packet to explore the loads and possible faults on each path before sending a larger flow, or remember proxies for common and recent destinations.

VI. EXPERIMENTS

A. Experimental setup

We compare up to five different routing algorithms for various $\mathcal{G}$-`Cell`s. They are: `DR`; shortest paths, computed by a breadth first search (`BFS`); `PR` with `GP_E`; `PR` with `GP_I`; and, `PR` with `GP_0`. Each routing algorithm (for a given DCN) is tested with the same $10,000$ input pairs, (src, dst). The estimated standard error of the mean is computed by $s_{\bar{x}}/\sqrt{trials}$, where $s_{\bar{x}}$ is the sample standard deviation and $trials = 10,000$. For our purposes of surveying the effects of different instances of `GP`, this value is negligible, and we therefore omit error bars in Figs. 2–3.

For each algorithm we plot $100(\bar{x}_{\mathtt{DR}} - \bar{x})/\bar{x}_{\mathtt{DR}}$ in Fig. 2, where $\bar{x}$ is the mean hop-length in the sample of computed routes. In other words, we plot the percent savings in hop-length over `DR`. Note that `GP_0` is implicitly plotted for $k = 2$ because it is equivalent to `GP_I` in this case.

We also plot, in Fig. 3, the mean number of proxies considered by `GP_I` and `GP_0` denoted $\bar{p}_{\mathtt{\_I}}$ and $\bar{p}_{\mathtt{\_0}}$, respectively, and the mean number of routes $\mathtt{PR}(src, dst)$ found to be no longer than $\mathtt{DR}(src, dst)$, denoted $\bar{r}_{\mathtt{\_I}}$ and $\bar{r}_{\mathtt{\_0}}$, respectively. Note that $\bar{p}\mathtt{\_I} = \bar{p}\mathtt{\_0}$ for $k = 2$ and, as such, this value is implicitly plotted for $k = 2$ in Fig. 3.

The two histograms in Fig. 4 show the proportion of links with a given load (number of flows) in β-DCell$_{3,3}$, under 1 million one-to-one communications, generated uniformly at random; one histogram is for `DR` and the other one is for `PR` with `GP_I`.

The networks we tested are given with their basic properties in Table II, and the details of each version of `GP`$*$ are given in Section V-A.

B. Evaluation

The plots in Fig. 2 show that for many $\mathcal{G}$-`Cell` topologies, significant savings in hop-length can be made over dimensional routes by using proxy routes, depending on the connection rule, network size, and the parameters k and n. It is immediate that `GP_I` and `GP_0` retain some good proxies, in relation to `GP_E`, which tries all of them. Furthermore, `GP_E` is comparable to `BFS`. Fig. 3 tells us how much searching each of the methods `GP_I` and `GP_0` must do, and how much path diversity they create, on average.

Note that the means plotted in Figs. 2–3 hide the success rate of `PR` in finding a good proxy path; as a typical example, $\mathtt{PR}(src, dst)$ is shorter than $\mathtt{DR}(src, dst)$ for approximately 30% of input pairs when using `GP_I` in DCell$_{3,6}$.

We highlight (and explain, where possible) some of the trends observable in the plot of Fig. 2: In general, proxy routes are more effective in β-DCell$_{k,*}$ than in DCell$_{k,*}$ and FiConn$_{k,*}$ of comparable size, with fixed k, however, even FiConn$_{k,*}$ still sees up to a 6–7$\%$ improvement.

The apparent weakness of `PR` in FiConn is partly explained by the fact that for given k and n, there are fewer proxy FiConn$_{m-1,n}$s to consider at level m. On the other hand we find that `GP_0` considers fewer than $g_1 = 6$ proxies for FiConn$_{3,10}$, while it considers more than $g_1 = 7$ proxies for DCell$_{3,6}$ and β-DCell$_{3,6}$. In addition, there are an equal number of potential proxy candidates in β-DCell$_{k,n}$ and DCell$_{k,n}$ in general, yet `GP_E`, `GP_I`, and `GP_0` invariably consider more proxy candidates for DCell$_{k,n}$, only to produce proxy paths that perform better in β-DCell$_{k,n}$. We must conclude that the connection rule and topology (FiConn vs Generalised DCell) profoundly impacts the performance of our proxy routing algorithms. This is somewhat unsurprising, however, since the connection rule and topology also affect the shortest paths; for example, the mean distance in β-DCell$_{3,3}$ is far shorter than in DCell$_{3,3}$ (see also [12]).

Proxy paths in larger networks (when increasing n) are worse than those in smaller networks, for each DCN with fixed k; for example DCell$_{3,3}$ and DCell$_{3,6}$, and also FiConn$_{3,10}$ and FiConn$_{3,16}$.

A related trend appears to be that for each family of DCNs, proxy-path-savings increase with k, in every version of `GP*`; for example, FiConn$_{3,10}$ and FiConn$_{4,6}$. The main reason for this is that the performance of `BFS`, relative to `DR`, also increases with k, thus providing a greater margin for improvement by using `PR`.

The difference between `GP_I` and `GP_0` grows with k (note that for $k = 2$, they are the same, and hence `GP_0` is not plotted for $k = 2$). This is because `GP_I` looks for sub-paths within a copy of $\mathcal{G}$-$\mathtt{Cell}_{k-2,n}$, whereas `GP_0` looks for sub-paths within a copy of $\mathcal{G}$-$\mathtt{Cell}_{0,n}$, and as the gap between 0 and $k-2$ increases, `GP_I` considers a larger set of proxy candidates. Similarly, we explain how the difference between `GP_E` and `GP_I` grows with k, but here it is the double exponential growth of $\mathcal{G}$-`Cell` that contributes extra

DCN	N	N/n	$\|E\|$	d	g_1	g_2	g_3
$F_{2,36}$	117648	3268	161766	7	19	172	
$F_{2,48}$	361200	7525	496650	7	25	301	
$F_{3,10}$	116160	11616	166980	15	6	16	121
$F_{3,16}$	3553776	222111	5108553	15	9	37	667
$F_{4,6}$	857472	142912	1259412	31	4	7	22
$F_{4,8}$	37970240	4746280	55768790	31	5	11	56
$D_{2,18}$	117306	6517	234612	7	19	343	
$D_{2,43}$	3581556	83292	7163112	7	44	1893	
$D_{3,3}$	24492	8164	61230	15	4	13	157
$D_{3,6}$	3263442	543907	8158605	15	7	43	1807

TABLE II
PROPERTIES OF THE DCNS IN OUR EXPERIMENTS. WE USE F TO ABBREVIATE FICONN, AND D TO ABBREVIATE (β-)DCELL.

proxy candidates to GP_E, since the search space for GP_I is proportional to g_{k-1}, whereas, GP_E considers exactly g_k proxy candidates (see Table II). Most notably, however, is the fact that for $\mathcal{G}$-Cell$_{2,*}$, the performance of GP_E is almost identical to the performance of GP_I; whereas DCell$_{2,43}$ has $g_1 = 44$, and $g_2 = 1893$, our results show that optimal proxies are nevertheless considered by GP_I (and hence, GP_0).

Although GP* is effective in computing shorter paths and comes fairly close to BFS (typically over 80% of the savings are obtained with PR), we can confirm that the shortest paths for these topologies are not, in general, a proxy route of the form we are considering in this paper as sometimes (e.g. (β-)DCell$_{3,3}$) this difference is considerable. This was expected, and provides motivation to explore novel general routing algorithms of order 3 and higher in future work.

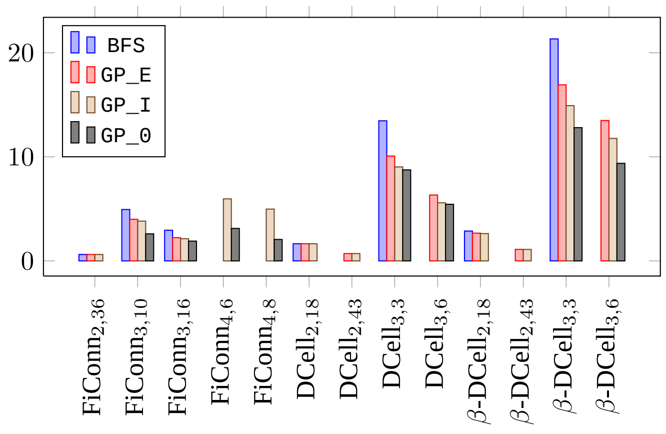

Fig. 2. Percent mean hop-length savings over DR.

Another benefit of proxy routing is that it also yields some path diversity which can be exploited for load balancing and fault-tolerance purposes. This can be seen in Fig. 3, where $\bar{r}$ is the number of distinct (but not necessarily disjoint) paths considered by PR(src, dst) that are no longer than DR(src, dst). Additional data must be studied, however, to determine exactly how $\bar{r}$ affects the load-balancing properties of the network.

We computed histograms that show the proportion of links with a given load, under 1 million one-to-one communications, plotted in Fig. 4. The histogram for GP_I is shifted left

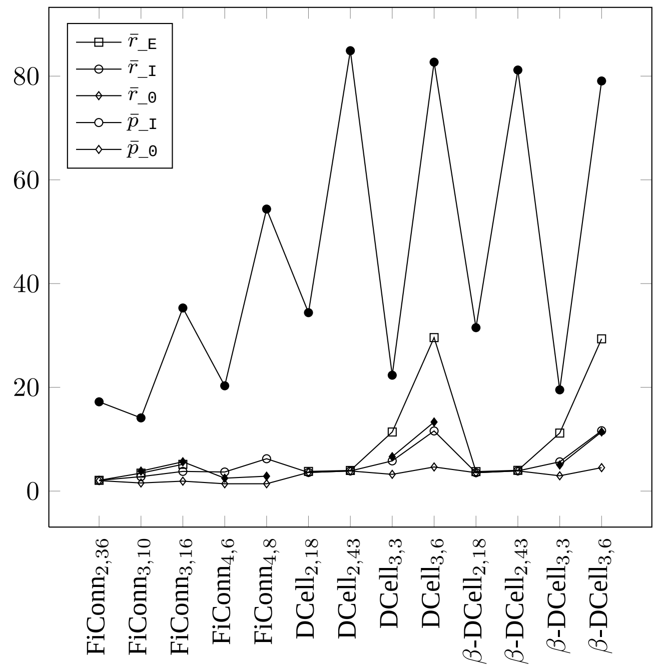

Fig. 3. Mean number of candidate proxies $\bar{p}$, and mean number of routes no longer than DR(src, dst), $\bar{r}$.

relative to the histogram for DR, meaning that many links carry less load than in the same scenario for DR. In addition, the maximum load is reduced (in our sample), suggesting many $\mathcal{G}$-Cells have a higher aggregate bottleneck throughput (ABT, introduced in [10], and closely related to the most heavily loaded link in the network) with PR than with DR.

Note that our primary focus is to reduce hop-length and implementation overheads of GP, and that we could increase path diversity even more if we were willing to route on longer paths than DR(src, dst); we do not do this here, but will explore this possibility in future research.

C. Significance

Various aspects of routing in a DCN depend heavily on the availability of short one-to-one paths. For example, minimising latency and energy usage, and building fault-tolerant and load balancing routing algorithms.

While there are inherent trade-offs in computing short

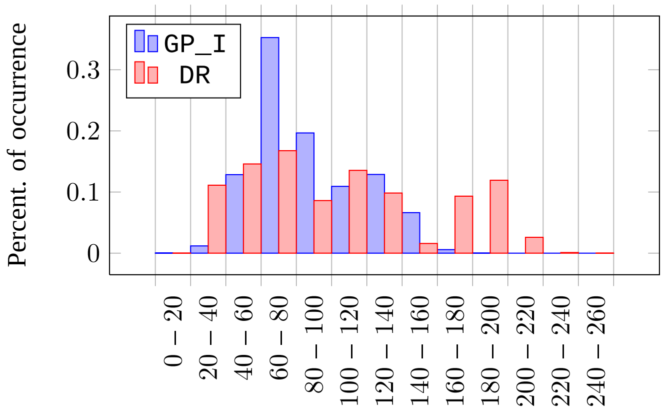

Fig. 4. Normalised histograms showing the proportion of links with a given load (number of flows), comparing `DR` with `PR` using `GP_I` β-DCell$_{3,3}$.

proxy routes, there are also multiple benefits: using shorter one-to-one paths in a DCN reduces the average latency of communications, the aggregate load, and thereby the energy usage; and, we obtain a non-deterministic path diversity at no extra cost while computing these paths, which can be used both adaptively or randomly to deal with faults and congestion, in addition to forming the building blocks of other fault-tolerant and load balancing routing algorithms (such as the way `DR` is used in `DFR` and `TAR`). As such, proxy routes are not only a good candidate for replacing `DR` in (Generalized) DCell and FiConn, they are also effective at performing some of the functions of the known adaptive routing algorithms for these networks, namely `DFR` and `TAR`, while simultaneously producing short paths.

VII. Conclusions and Future Research

In this paper we have shown that the topologies of the DCNs Generalized DCell and FiConn are completely connected recursively-defined networks. As such, we characterised all possible routes (with no repeated nodes) on these networks and then proposed the family of routing algorithms `PR` to compute proxy routes; that is, general routes of order 3. We detailed three instances of this family, `GP_E`, `GP_I`, and `GP_0`, where each one considers a number of candidate proxy sub-structures, and selects the optimal proxy to route through. We performed an analytical and empirical comparison between these, shortest paths, and the previously known dimensional routes, as regards mean hop-length; The main results of our experiments are that significant savings in hop-length can be made over dimensional routes by using proxy routes, even with only a relatively small set of candidate proxies, and that the amount of savings depends on connection rule, network size, and the parameters k and n.

In future research we will perform a deeper analysis of the DCNs in question, with two major goals. The first one, motivated by the fact that `GP_I` sometimes discards the optimal proxy candidate, calls for a closer inspection of the topologies. We want to both find the optimal proxy candidates, and reduce the size of the search space.

Furthermore, whereas this paper is focused on dimensional and proxy routing, there may be cases where no shortest path between two server-nodes is a dimensional route or a proxy route. Note that whilst a given shortest path may be found not to be a dimensional or proxy route, this does not preclude other paths with the same terminal nodes from being dimensional or proxy routes. A deeper mathematical analysis of the DCNs in question may shed light on (1) whether or not higher-order routing algorithms are needed, and (2) how to compute optimal routes of this type efficiently.

Acknowledgement This work has been funded by the Engineering and Physical Sciences Research Council (EPSRC) through grants EP/K015680/1 and EP/K015699/1.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*. New York, NY, USA: ACM, 2008, pp. 63–74.

[2] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2012.

[3] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: A cost-efficient topology for high-radix networks," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 126–137, Jun. 2007.

[4] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, Aug. 2010.

[5] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A scalable and fault-tolerant network structure for data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 75–86, Aug. 2008.

[6] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, S. Lu, and J. Wu, "Scalable and cost-effective interconnection of data-center servers using dual server ports," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 102–114, 2011.

[7] D. Guo, T. Chen, D. Li, M. Li, Y. Liu, and G. Chen, "Expandable and cost-effective network structures for data centers using dual-port servers," *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1303–1317, 2013.

[8] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 51–62, Aug. 2010.

[9] Y. Liao, J. Yin, D. Yin, and L. Gao, "DPillar: Dual-port server interconnection network for large scale data centers," *Computer Networks*, vol. 56, no. 8, pp. 2132–2147, May 2012.

[10] 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," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, Aug. 2009.

[11] M. Kliegl, J. Lee, J. Li, X. Zhang, C. Guo, and D. Rincon, "Generalized DCell structure for load-balanced data center networks," in *INFOCOM IEEE Conference on Computer Communications Workshops*, 2010, pp. 1–5.

[12] M. Kliegl, J. Lee, J. Li, X. Zhang, D. Rincon, and C. Guo, "The generalized DCell network structures and their graph properties," October 2009, Microsoft Research.

[13] R. Diestel, *Graph Theory, 4th Edition*, ser. Graduate Texts in Mathematics. Springer, 2012, vol. 173.

[14] G.-H. Chen, S.-C. Hwang, M.-Y. Su, and D.-R. Duh, "A general broadcasting scheme for recursive networks with complete connection," in *Proceedings of the 1998 International Conference on Parallel and Distributed Systems*, Dec 1998, pp. 248–255.

[15] G. D. Vecchia and C. Sanges, "A recursively scalable network VLSI implementation," *Future Generation Computer Systems*, vol. 4, no. 3, pp. 235–243, 1988.

[16] D.-R. Duh and G.-H. Chen, “Topological properties of WK-recursive networks,” *Journal of Parallel and Distributed Computing*, vol. 23, no. 3, pp. 468–474, 1994.