


# The influence of datacenter usage on symmetry in datacenter network design

Iain A. Stewart<sup>1</sup>  · Alejandro Erickson<sup>1</sup>

Published online: 16 December 2017

© The Author(s) 2017. This article is an open access publication

**Abstract** We undertake the first formal analysis of the role of symmetry, interpreted broadly, in the design of server-centric datacenter networks. Although symmetry has been mentioned by other researchers, we explicitly relate it to various specific, structural, graph-theoretic properties of datacenter networks. Our analysis of symmetry is motivated by the need to ascertain the usefulness of a datacenter network as regards the support of network virtualization and prevalent communication patterns in multi-tenanted clouds. We argue that a number of structural concepts relating to symmetry from general interconnection networks, such as recursive-definability, the existence and dynamic construction of spanning trees, pancyclicity, and variations in Hamiltonicity, are appropriate topological metrics to use in this regard. In relation to symmetry, we highlight the relevance of algebraic properties and algebraic constructions within datacenter network design. Built upon our analysis of symmetry, we outline the first technique to embed guest datacenter networks in a host datacenter network that is specifically oriented towards server-centric datacenter networks. In short, we provide the graph-theoretic foundations for the design of server-centric datacenter networks so as to support network virtualization and communication patterns in cloud computing.

**Keywords** Datacenter networks · Server-centric · Network topology · Virtualization · Communication patterns · Topological metrics · Graphs

---

The research in this paper was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) Grant EP/K015680/1 ‘Interconnection Networks: Practice unites with Theory (INPUT)’.

---

✉ Iain A. Stewart  
i.a.stewart@durham.ac.uk

Alejandro Erickson  
alejandrorickson@gmail.com

<sup>1</sup> Department of Computer Science, Durham University, South Road, Durham DH1 3LE, UK

## 1 Introduction

The primary aim of this paper is to show how certain structural graph-theoretic properties, characterized as *symmetry in datacenter networks*, can be used to evaluate aspects of usage and communication in datacenters. This paper is both the first significant effort to define symmetry in datacenter networks, and the first to relate symmetry to important datacenter properties such as virtualization, as it relates to cloud computing, and communication patterns, arising from distributed data processing. As such, this paper opens up the study and design of datacenters and their networks to a wider theoretical community, within which lies a significant capacity to contribute to what is rapidly becoming the pervasive global computing infrastructure.

In Sect. 1.1 of this introduction, we explain some essential concepts relating to the server-centric datacenter networks studied in this paper, before highlighting how network topology designs are traditionally evaluated in Sect. 1.2. In Sect. 1.3, we give examples of datacenter usage that have hitherto not impacted upon datacenter network design, and in Sect. 1.4, we overview the various research contributions made in this paper.

### 1.1 Datacenter network design

Datacenters are used for a variety of purposes such as large-scale search, e-commerce, media hosting and distribution, social networking, and large-scale scientific and data-intensive applications (such as weather forecasting, healthcare, bioinformatics, and data mining). Datacenters can be dedicated to one specific use under the control of one owner, e.g. a Facebook datacenter under the control of Facebook. Alternatively, and as is increasingly the case as cloud computing gathers pace, datacenters can incorporate multiple concurrent applications under the control of multiple users, each of whom has shared access to a portion of the overall datacenter, e.g. Amazon's Elastic Compute Cloud. The reader is referred to a plethora of review articles for a general background as regards datacenters and issues only touched upon or not mentioned here (e.g. [1, 5–7, 34, 41, 51, 53, 59, 63, 64]).

The prevailing architecture is that datacenters consist of tree-like fabrics of switches with the servers as terminals within this fabric, e.g. Fat-Tree [3], VL2 [28], and Portland [52]. It is generally acknowledged that this basic *switch-centric* architecture, where routing is handled by routing tables within the switches (and so there can be no direct server-to-server links), has its limitations as datacenters grow (see, for example, [51]). New datacenter architectures have subsequently emerged which include the *server-centric* architecture whereby interconnection intelligence is imposed on the servers with switches only acting as 'dumb' crossbars; now there can be server-to-server and server-to-switch links, but not switch-to-switch links. Typical examples of such server-centric datacenter networks (DCNs) are DCell [31], BCube [29], FiConn [44], DPillar [49], HCN and BCN [32], SWCube, SWKautz, and SWdBuijn [45], and GQ\* [22], and it is with server-centric DCNs that we are concerned exclusively in this paper.

The construction of an actual datacenter is a mammoth and massively expensive task with some datacenters reputed to have hundreds of thousands of servers; moreover, researchers are currently exploring building datacenters incorporating millions

of servers. Of course, the design of a datacenter is painstakingly undertaken so that a good understanding of the efficacy of the eventual datacenter can be obtained without having to build it. At the root of the design of a datacenter is the design of the DCN's physical topology; that is, the topology of how the servers and switches are all interconnected. There are many dimensions involved in DCN topology design such as (incremental) scalability, latency, wiring complexity, bisection bandwidth, connectivity, (aggregated) throughput, routing algorithms, fault tolerance, energy efficiency, costs and packaging, traffic patterns, reliability, security, and agility (see [59] for a survey of many of these aspects). Of course, some of these aspects work against each other and DCN topology design is a challenging arena where trade-offs have to be made.

Henceforth, for brevity, we write 'DCN' to mean 'DCN topology', and we abstract a server-centric DCN as an undirected graph where the nodes are partitioned into *server-nodes* and *switch-nodes*. Notice that the partition of the nodes of a graph abstracting a DCN into server-nodes and switch-nodes, in tandem with the entirely different functionalities of servers and switches in datacenters, makes a graph-theoretic abstraction of a DCN different from the usual notion of a graph.

## 1.2 Evaluation of designs

There are three essential mechanisms by which the design of a general interconnection network can be evaluated: by building a small prototype; by simulation; and by investigating what might be called topological metrics (that is, beneficial structural properties). Given that DCNs aim to consist of hundreds of thousands of servers and switches, building a small prototype is simply not effective; behaviours within a 500,000 server datacenter will likely not manifest themselves in a small test bed of, say, 30 servers. Besides, even were a test bed to be helpful, researchers must ascertain whether building one is worth the cost in terms of finance and labour (indeed, many academic researchers do not have the financial capacity to build a test bed). That leaves simulation and topological metrics as the primary means to predict the real-world performance of DCNs.

Topological metrics are essentially used to advise on the likely behaviour of the constructed interconnection network with regard to some specific issue of interest, such as the way path length relates to latency and connectivity to fault tolerance. Experience has shown, particularly in the distributed-memory multiprocessor environment, that many of the metrics used truly reflect eventual behaviour. As such, the use of metrics is crucial as a foundational tool, with simulation providing empirical confirmation. It is with topological metrics, on the graph-theoretic abstraction of a DCN, that we are primarily concerned in this paper.

The design of interconnection networks for distributed-memory multiprocessors and networks-on-chips is well established (see, for example, [15], which is the standard text, particularly as regards distributed-memory multiprocessors, and [40], where the focus is on networks-on-chips) and a thriving area of research. Whilst the design of DCNs is more recent, it has much in common with general interconnection network design yet there are profound differences too, prompted by, for example, usage, scale, and packaging. Hitherto, the most common metrics used for DCN evaluation

are the availability of routing algorithms, hardware cost (e.g. number of servers and switches), hardware complexity (e.g. number of server ports), diameter, bisection width, connectivity, and shortest-path lengths. It is probably fair to say that the development of appropriate topological metrics for DCNs is not as advanced as it is for distributed-memory multiprocessors and networks-on-chips, and that the validity of these topological metrics within a datacenter environment is not as well established. Our paper seeks to strengthen the role of topological metrics in DCN design.

### 1.3 Datacenter usage

In this paper, we consider some distinctive uses to which datacenters are put, so as to differentiate them from distributed-memory multiprocessors and networks-on-chips. Our aim is to examine these uses in some detail so that we might develop appropriate topological metrics that reflect beneficial properties of datacenters with regard to this usage. This paper can be seen as one of advocacy for a more systematic consideration of datacenter usage as informing datacenter design; as such, it is perhaps the first paper to do this in any great detail. The aspects of datacenter usage that we consider, from amongst many possibilities, are virtualization and the implementation of communication patterns.

Virtualization is a fundamental concept in cloud computing where virtual machines and virtual datacenters, belonging to users, are embedded at physical locations within the host datacenter so that quality of service guarantees can be given to the users, but also so as to secure an efficient embedding from the perspective of the cloud owner. We examine virtualization through the lens of SecondNet [30] and Oktopus [4], two mechanisms for manipulating virtual machines and virtual datacenters within the datacenter hosting a cloud. There is nothing particularly special about our choice: both SecondNet and Oktopus are representative of (and influential in) the current virtualization landscape and simply allow us to examine virtualization in a concrete context so as to draw topology-based conclusions from their usage. Whilst the intended targets of SecondNet and Oktopus are switch-centric DCNs (indeed, as yet no virtualization methodology has been developed specifically for server-centric DCNs), we examine the general principles behind SecondNet and Oktopus and consider these principles within a server-centric context.

In addition to virtualization, the different uses to which datacenters are put result in a variety of communication patterns needing to be supported within distributed data processing. Just as we used SecondNet and Oktopus to provide a concrete context for virtualization, we use MapReduce [16, 17] to do likewise as regards (primarily) many-to-many communication patterns.

### 1.4 Our contributions

There are various novel aspects to our research contributions.

- We undertake the first study of how datacenter usage can influence server-centric DCN design, through a detailed examination of existing virtualization

- methodologies applicable to switch-centric DCNs and of potential support for (many-to-many) communication patterns which feature in datacenter applications.
- We outline the first generic virtualization methodology for application within server-centric DCNs (the omission of such a consideration was noted in [5] and suggested as a direction for further research; in so far as we are aware, our methodology is the first to be proposed).
  - We show that support for datacenter usage and intrinsic communication patterns is closely correlated with certain topological aspects of DCNs that we identify as *symmetry*. We define symmetry in DCNs broadly, encompassing structural properties that are applied locally, but universally within any locality; basic illustrations might be that every switch has the same number of ports, every server sits on a cycle of servers of some given length, or every server is part of a sub-DCN that is similar to the whole. We show that concepts such as recursive-definability, complete connection, pancyclicity, variations in Hamiltonicity, and other generic algebraic constructions all provide topological support (so, we add weight to the general acceptance of these symmetry-based metrics in interconnection design). We rapidly come to the conclusion that DCNs should be viewed compositionally as topologies within which other topologies have multifarious embeddings.
  - We explain how the established notions of symmetry in interconnection networks, that is, node-symmetry and link-symmetry, are perhaps not so relevant to DCNs and that other notions of symmetry are more pertinent. Although other researchers have mentioned symmetry within a DCN, insofar as we are aware our paper contains the first systematic analysis of different aspects of symmetry within a server-centric DCN context. We use the DCN HCN (from [32]) to refine and exemplify our discussions of symmetry (we provide the definition of HCN in Sect. 5), but we also mention other existing server-centric DCNs in the context of symmetry (albeit more briefly).
  - We highlight the efficacy of using existing interconnection network research to support the design of DCNs and also demonstrate how the use of algebraic methods is beneficial to the design process.

It is important to note that our aim is not to be definitive as to what makes a DCN ‘properly symmetric’, but to argue as to why various aspects of symmetry are very relevant as regards datacenter design in relation to how datacenters are used. In short, we provide the theoretical foundations for the design of server-centric DCNs so that the usage of DCNs, through support for virtualization and communication patterns, is a primary concern.

Our work sits between the engineering process of building datacenters and the theoretical consideration of abstractions of DCNs as discrete structures; that is, it is graph theory targeted towards a practical application area. Our intention throughout is: to provide enough information concerning server-centric DCNs and their design for the theoretician to appreciate the underlying practical issues that go to influence any graph-theoretic abstraction and analysis; and to demonstrate that theoretical ideas and concepts are relevant to and can impact significantly on the study of DCNs. As such, our paper provides an introduction to the design of server-centric DCNs for

graph theorists who, we hope, can consequently contribute to the development of the computational infrastructures that underpin cloud computing.

There is a long path from the theoretical formulation of the server-centric DCN paradigm to the construction of real, large-scale server-centric datacenters; this path is made longer due to the significant costs of actually building such a datacenter and the need to be sure that what emerges will be fit for purpose in an engineering sense. Our work is novel in that it lies one step further along this path than most current server-centric DCN research: it is motivated by the applications that run on datacenters and the traffic patterns that emerge, rather than just internal aspects of routing within DCNs. We have more to say about the subsequent development of our research and framework in our conclusions.

## 1.5 The structure of this paper

In the next section, we detail some basic definitions and concepts before looking at datacenter virtualization and communication patterns, and deriving some preliminary influences of this usage on design, in Sect. 3. In Sect. 4, we consider symmetry in a broad sense and within interconnection networks in general. In Sect. 5, we define the DCN HCN before undertaking a systematic consideration of aspects of symmetry in relation to virtualization and communication patterns in Sect. 6. Our analysis in Sect. 6 feeds into Sect. 7 where we develop the first virtualization methodology geared towards server-centric DCNs. Our conclusions and directions for further research are given in Sect. 8.

## 2 Basic definitions and concepts

We abstract DCNs as (undirected) graphs such that nodes are either *server-nodes* or *switch-nodes*. Switch-nodes are used to interconnect groups of server-nodes and direct server-node to server-node links are also allowed; as such, we are dealing with *server-centric* DCNs. We reiterate that we do not allow switch-node-to-switch-node links as to do so would require interconnection intelligence at the switches modelled by our switch-nodes; we insist that switches should operate only as ‘dumb’ crossbars. The upshot is that our server-centric DCNs are abstracted as graphs where the node set is partitioned into a set of server-nodes and a set of switch-nodes. We also refer to our DCNs as *topologies* (when we are emphasizing the interconnection patterns).

The notion of a graph embedding will prove to be important in our work. Graph embeddings are well established within the study of interconnection networks (see, for example, [61, Sect. 1.3.2] for more details).

**Definition 1** Let  $H = (U, F)$  be the guest graph and let  $G = (V, E)$  be the host graph. A *graph embedding* is a mapping  $f: U \rightarrow V$  so that, in addition, every link  $(x, y) \in F$  is explicitly associated with a path joining  $f(x)$  and  $f(y)$  in  $G$ ; call these paths  $f(F)$ . The *dilation* of the embedding is the maximum length of any path in  $f(F)$ ; the *congestion* of the embedding is the maximum over all links  $e \in E$  of the number

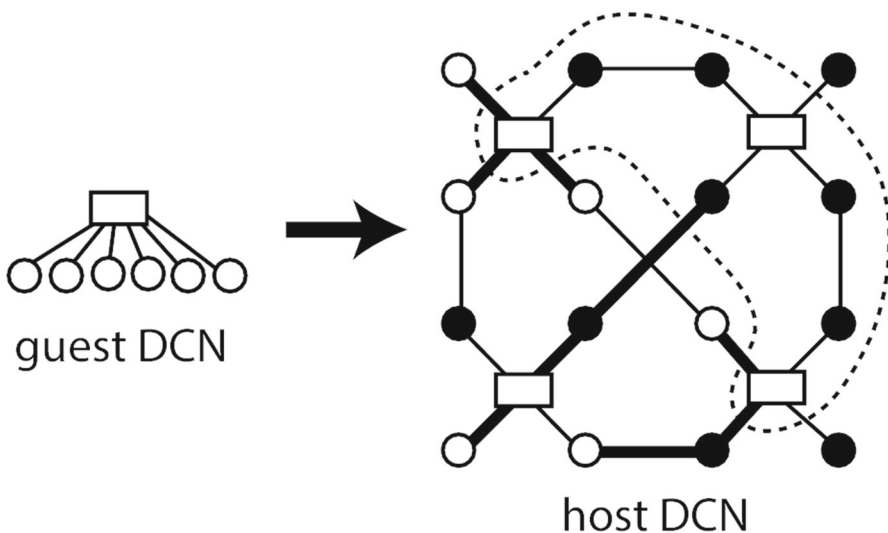
of paths in  $f(F)$  on which  $e$  lies; and the *load* of the embedding is the maximum over all nodes  $x$  of  $G$  of the total number of nodes of  $H$  that are mapped to  $x$  by  $f$ .

The parameters dilation, congestion, and load have clear relevance as regards, for example, the simulation of one distributed-memory multiprocessor by another (with, of course, an optimal embedding being one where all of these parameters have the value 1).

Our notion of an embedding within a datacenter context will be slightly different and more flexible than that in Definition 1; for one thing, we have server-nodes and switch-nodes to consider. Our embeddings map guest DCNs to host DCNs so that: server-nodes are mapped to server-nodes; switch-nodes are mapped to (connected) sub-networks within the host DCN; and links are mapped to paths of links, and so that the ‘virtual’ DCN induced by the images of the server-nodes, switch-nodes, and links under the embedding is connected. The notions of load, dilation, and congestion for a DCN embedding can be amended appropriately (we do not provide these details here as throughout this paper we are only concerned with general principles relating to embeddings and not precise analytical measurements).

An example of an embedding of a switch-node with 6 adjacent server-nodes in a DCN can be visualized as shown in Fig. 1 (the server-nodes are circles and the switch-nodes rectangles) where: the sub-network surrounded by a dotted line in the host DCN corresponds to the switch-node in the guest DCN; the white circles correspond to the server-nodes of the host DCN to which the server-nodes of the guest DCN have been mapped; and the bold (paths of) links in the host DCN correspond to the server-node-to-switch-node links in the guest DCN.

We mention a number of DCNs and interconnection networks in what follows. The precise definitions of these DCNs can be found in: [31] for DCell; [29] for BCube; [44] for FiConn; [49] for DPillar; [44] for HCN and BCN; and [45] for SWCube, SWKautz,



**Fig. 1** An embedding of one DCN in another

and SWdBruijn. The references [15,37,61] provide details of other interconnection networks and graph-theoretic concepts we happen to mention in this paper.

We end with a remark on terminology. Strictly speaking, an interconnection network is a family of networks, parameterized by at least one parameter; for example, there is a hypercube of dimension  $n$ , for each positive integer value of  $n$ . The same can be said of DCNs. However, for brevity, we usually refer to such a family of interconnection networks as ‘an interconnection network’, and likewise for families of DCNs. Hence, when we write, for example, ‘the DCN HCN’, what we really mean is ‘the family of DCNs HCN’.

### 3 Datacenter usage

We now examine two aspects of datacenter usage that should influence the design of DCNs: virtualization and the implementation of communication patterns. Our intention in this section is to draw out issues within virtualization and communication patterns that directly impact upon DCN design. As we shall ultimately see, (various notions of) symmetry within a DCN can support both virtualization and the implementation of communication patterns.

#### 3.1 Virtualization

Three essential services are provided by cloud providers (see, for example, [63]): IaaS, or Infrastructure as a Service, provides (possibly shared hardware) allocations of the cloud to users (or tenants); PaaS, or Platform as a Service, provides, for example, operating system support and software development frameworks to users; and SaaS, or Software as a Service, provides on-demand applications to users. An important aspect of datacenter support for IaaS is virtualization. *Virtualization* is a mechanism by which datacenter hardware (the servers, switches, links, etc.) is both shared and conglomerated so as to form virtual machines and virtual datacenters to be made available to users for rent. A virtual machine might reside at a server, but share the processor(s) with other virtual machines residing at the same server. Virtual machines, possibly residing at different processors, might be joined via virtual links (which are essentially paths of real links, within the host datacenter) and/or virtual switches (which are essentially sub-networks of real switches, servers, and links) so as to form virtual datacenters. Users can pay rent for virtual machines or virtual datacenters within the datacenter forming the cloud (that is, the host datacenter) and possibly also for guarantees of quality of service. A user request to the cloud owner might be to provide a virtual datacenter with a user-specified topology and dedicated stipulated link bandwidth, computational power, memory, storage requirements, and so on. In a graph-theoretic sense, virtualization is a (much) more complex version of graph embedding.

In order to support virtualization, a datacenter owner needs to be able to not only embed guest topologies within the host DCN, but also to arrange that these embeddings come with the requisite amounts of resource and that different such embeddings can all ‘fit together’ within the host. It is clear how DCN analogies of concepts such as



dilation, congestion, and load, from Definition 1, will have relevance. Importantly, embeddings need to be:

- flexible, in that there should be a variety of possibilities available so that a virtual datacenter can be physically located in a beneficial location depending upon, for example, other current virtual datacenter embeddings, hardware usage, and traffic loads (beneficial in that the owner's physical resources are used efficiently and the tenant's performance guarantees are achievable);
- identifiable, in that there should be a mechanism by which the different possibilities for (physical) location can be efficiently generated by the datacenter owner; and
- agile, in that there should be a mechanism for the migration of existing embeddings to other parts of the host datacenter so as to free up the previously tied resources for subsequent embeddings.

Whilst we focus on graph-theoretic embeddings (given that our aim is to find structural properties of DCNs that support such embeddings, and so virtualization), it should not be forgotten that associated with these embeddings in reality are resource demands regarding link bandwidths, CPU speeds, memory, server bandwidth, and so on.

As a simple illustration as to the additional complexity that this can impose, suppose that two virtual machines  $m_1$  and  $m_2$  reside at a server and two virtual machines  $n_1$  and  $n_2$  reside at another server so that a virtual link joining  $m_1$  and  $n_1$  forms a virtual datacenter with a virtual link joining  $m_2$  and  $n_2$  forming another virtual datacenter, where these virtual links share the same actual physical path of links joining the servers in the host datacenter. The bandwidth of this physical path of links needs to be shared between the two virtual datacenters; moreover, the actual physical bandwidth needs to be enough to accommodate the required bandwidths for the two virtual links (of course, the path links might also be currently used by other virtual datacenter embeddings). Implementing full-blown virtualization is incredibly difficult to achieve and is a vibrant area of research, with embedding just one aspect of virtualization (albeit a significant one); there are many other (more practical) aspects to consider too such as scalability, fault tolerance, security, performance isolation, and monitoring (the reader is referred to recent surveys [5,56,60] for detailed accounts of the current state of the art as regards datacenter virtualization).

In order that we might get a flavour of how heuristic methods are applied to embed virtual datacenters within host datacenters (and so obtain an appreciation of the sort of structural properties of DCNs that might support virtualization), let us take a brief look at two influential methods: SecondNet [30] and Oktopus [4]. These two methodologies are good illustrations of the current research landscape as regards datacenter virtualization. They are primarily geared towards switch-centric DCNs, but the basic methodologies can be applied within server-centric DCNs too. Both SecondNet and Oktopus are more relevant to server-centric DCNs than methodologies such as FlowVisor [54] and FlowN [18]: SecondNet and Oktopus enforce virtual datacenter isolation through hypervisors installed at servers, whereas FloVisor and FlowN enforce isolation through configured rules at switches (however, we have more to say about the applicability of SecondNet and Oktopus to server-centric DCNs presently). We emphasize

that we are only interested in the embedding algorithms underlying SecondNet and Oktopus, and that there are many other more practical aspects to both.

### 3.1.1 *SecondNet*

In SecondNet, the focus is on providing bandwidth guarantees for a set of virtual machines between each pair of which there is a required bandwidth constraint. In a sense, virtual datacenters in the form of cliques are being embedded though other topologies can be embedded by setting the bandwidth requirement of specific virtual links to 0. The embedding algorithm has the limitation that distinct virtual machines must be allocated to distinct physical servers, and it proceeds as follows. Core to the methodology is that the host datacenter can be partitioned into (not necessarily disjoint) sub-networks of varying numbers of (interconnected) servers.

Given some virtual datacenter consisting of  $m$  virtual machines, say, a sub-network of at least  $m$  servers in the host datacenter is sought. In the first phase of the algorithm, a bipartite graph is constructed with nodes representing the  $m$  virtual machines on one side of the partition and nodes representing the servers of the host sub-network on the other. There are edges introduced into this bipartite graph to denote the possible embedding of a virtual machine at a server (this is subject to CPU, memory, server bandwidth, and other requirements) and edge weights are added reflecting the used (ingress and egress) server bandwidth of embedding the given virtual machine at the given server. A min-cost network flow algorithm is used to obtain a minimum weight matching where every virtual machine is matched with some server, if one exists; if one does not exist then the search within this sub-network is terminated and we begin again with a different sub-network.

In the next phase, if a matching has been found, then the algorithm searches for paths between the servers within the sub-network so that the virtual link bandwidth requirements of the different pairs of virtual machines are accommodated (of course, there is nothing to be done if this bandwidth requirement is 0). This search is on a greedy, path-by-path basis (working through the virtual links in order of decreasing required bandwidth) using a simple shortest-path algorithm and amending link bandwidth availability as the algorithm proceeds. If such a set of paths cannot be found, then the search within this sub-network terminates and we begin again with a different sub-network. If a set of paths can be found, then the embedding is made, with all residual capacities in the host datacenter amended to reflect the embedding, and the host datacenter is ready to receive another virtual datacenter for embedding.

The actual allocation of some virtual datacenter is undertaken by a centralized manager, hosted at some server, that communicates with other servers in the datacenter via a signalling spanning tree (that can be evolved in the case of node or link failures). We reiterate that key to the philosophy of SecondNet is that the host servers can be grouped into sub-networks of different sizes with these sub-networks iteratively explored as regards to whether some virtual datacenter can be embedded. There is also scope for amending existing virtual datacenter embeddings and also migrating existing embedded virtual machines (primarily so as to defragment the allocation of virtual datacenters with respect to the sub-networks).

### 3.1.2 *Oktopus*

In *Oktopus*, the focus is on providing bandwidth guarantees for virtual datacenters in the form of virtual machines all connected to a virtual switch, called *virtual clusters* in [4], and also collections of virtual clusters where the virtual switches of these clusters are connected to a root virtual switch, called *oversubscribed virtual clusters* in [4] (as is noted in, for example, [14], the Amazon EC2 cloud embeds tenant requests in the form of a virtual cluster). It is remarked in [4] that other virtual datacenters in the (topological) form of hypercubes, multidimensional meshes, de Bruijn networks, and so on might also be offered to users (though this possibility is not seriously examined). Crucial to the ethos of *Oktopus* is that hierarchical, recursively defined, tree-like topologies are key, both as regards the virtual datacenters to be embedded and the host datacenter.

The embedding algorithm proceeds (roughly) as follows. It initially only considers host servers and tries to embed the given virtual cluster or oversubscribed virtual cluster entirely within each server (as to whether the virtual datacenter can be so embedded depends solely on whether the server can support the number of virtual machines in the virtual datacenter). If unsuccessful, then the algorithm looks for an embedding at the next ‘level’ up. This has the effect of looking for an embedding within each sub-network of the host datacenter consisting of a number of servers and their parent switch, so that link bandwidth constraints are maintained (each such grouping within the host is considered). If unsuccessful, the algorithm looks for an embedding at the next level up which means looking within each sub-network formed by servers, parent switches of servers, and a parent switch of these switches. This continues until either an embedding is found or none is possible. In a sense, there is commonality with *SecondNet* in that a sequence of sub-networks is iteratively explored; here, the sub-networks are determined by the tree-like structure of the host topology.

Like *SecondNet*, there is a centralized network manager within *Oktopus* that maintains the current situation as regards which virtual clusters and oversubscribed virtual clusters are currently embedded where along with the residual link bandwidths and server capacities available. Also, it is stated in [4] that *Oktopus* can be extended to deal with the migration of already embedded virtual clusters and oversubscribed virtual clusters.

Both *SecondNet* and *Oktopus* are symptomatic of current methodologies: almost all are primarily targeted towards switch-centric DCNs, such as *Fat-Tree* [3], *VL2* [28], and *Portland* [52], or server-centric DCNs that are heavily tree based such as *BCube* [29], and they are heuristic based. It is immediately apparent that *Oktopus* is strongly geared towards tree-based topologies; however, *SecondNet* is too with its clustering principles based around racks and pods (see [30]; of course, any tree-based methodology can be used in an arbitrary DCN simply by utilizing spanning trees within sub-networks). Having said this, *DCell* is mentioned in [30] as a possible DCN to which *SecondNet* might be applied, but this is more in terms of *DCell*’s routing capabilities rather than with regard to support for embedding. The simulations undertaken for *SecondNet* in [30] are done so in *Fat-Tree*, *VL2*, and *BCube* (with the notion of sub-network derived from the tree-based structure of the host topology), and the simulations undertaken for *Oktopus* in [4] are done so in simple three-level tree

topologies. Centralized control in both SecondNet and Oktopus is undertaken via a spanning tree though this would probably also be the case in any datacenter (if there is to be a solitary network manager server). In summary, virtualization has not yet been seriously considered for the (non-tree-based) server-centric datacenters of interest to us and the structural analysis for doing so is as yet undeveloped.

Both SecondNet and Oktopus are also symptomatic of another aspect of current approaches to virtualization: virtualization (and embedding) is generally undertaken independently of the actual underlying host DCN (to some extent this is understandable given that the DCNs to which these methodologies are applied are all tree based and it is this structural property that dominates the methodologies). On the one hand, both SecondNet and Oktopus can be applied within a variety of (tree-based) datacenters; on the other hand, the actual topology of a DCN has not been fully utilized. It would appear that there might be scope for a better use of the actual host topology in virtualization tools such as SecondNet and Oktopus. This remark is also made in [5] where it is noted that the network utilization of SecondNet varies according to the underlying host topology; indeed, the authors of [5] remark upon the current lack of a consideration of embeddings on DCN design and mention this topic explicitly as a future research direction.

The general approach taken in very recent work on virtualization (which builds on SecondNet and Oktopus) is as follows: the underlying DCN topology is ignored; it is noted that the general virtualization problem is **NP-hard**; and solutions are developed based on heuristic methods, e.g. ant colony optimization algorithms in [66], greedy algorithms in [48], and linear programming in [14, 65]. Of course, with a fixed DCN as the host, it is possible that the virtualization problem becomes solvable in polynomial time (though, in our view, unlikely for existing DCNs). In any case, graph-theoretic properties of the host DCN topology have so far not been used.

Within this paper, it is our intention to identify structural aspects of DCNs that are conducive to virtualization; however, we also outline a new methodology for developing virtualization algorithms for server-centric DCNs. Even though our virtualization methodology uses the underlying host DCN topology, it is not tied to one particular DCN as it actually only uses structural properties of the DCN that are prevalent in many server-centric DCNs. Thus, just as existing tree-based methodologies are widely applicable, so is ours within the landscape of server-centric DCNs. We shall return to our discussion of datacenter virtualization and current methodologies when we present our own server-centric virtualization methodology in Sect. 7.

### 3.2 Implementing communication patterns

The actual use a datacenter, or a virtual datacenter, is put to can result in specific communication patterns being prevalent. We are heavily influenced by MapReduce (see, for example, [16, 17]) which is extremely common within many distributed data-intensive applications and which gives rise to many-to-many traffic patterns. In short, MapReduce has three essential phases: a map phase; a shuffle phase; and a reduce phase.

- In the map phase, a master server assigns a map task to a collection of worker servers (the mappers) at which the inputs for these tasks are stored. Each mapper produces intermediate data in the form of key–value pairs.
  - For example, each mapper might have local access to a file of text and the map task might be for each mapper to compute the number of occurrences of each word in its text file. Each key of a key–value pair is an actual word, and each value is the number of occurrences of that word in the text file.
- On completion of the map tasks, in the shuffle phase the mappers redistribute the intermediate data based upon the keys so that after the shuffle phase all data corresponding to some key reside at the same worker server (the reducers); alternatively, the mappers might send location details of their intermediate data to the master server.
  - In our example, after the shuffle phase, all data relating to the same word resides at the same reducer.
- In the reduce phase, each reducer executes its reduce task on the intermediate key–value data residing at that server; alternatively, the master server distributes location data to the reducers with each reducer given a key or set of keys so that each of these reducers then reads the intermediate values corresponding to its key or keys from (possibly) other reducers and executes its reduce task.
  - In our example, the reduce task might be for each reducer to sum all of the instantiations of keywords that have been assigned to it.

It could well be that in a virtual datacenter (or indeed the whole datacenter) every server plays the role of both a mapper and a reducer. Clearly, with MapReduce, communication patterns such as many-to-many, one-to-many, and many-to-one need to be supported by the underlying topology (see, for example, [62]). We focus in this paper on many-to-many as it is the most important communication pattern as regards MapReduce (and generally the most difficult to implement).

Other usage examples, such as data replication (whereby chunks of data are replicated at various servers, so as to aid search query latency, for example), distributed file systems in general (such as that in [27]), and Web searching, also give rise to one-to-many and many-to-one communication patterns (see, for example, [11,62]). There is a detailed consideration of multicast, or one-to-many communication, in datacenters in [46].

### 3.3 The basic influence of usage

The necessity for virtualization in datacenters means that unlike the situation for distributed-memory multiprocessors and networks-on-chips, we should not necessarily be looking at the DCN in its entirety; we should be looking at the DCN as being composed of constituent and interlinked parts. We should not necessarily demand that the whole DCN possesses some topological property, for example, but that within the DCN there are (numerous) sub-networks possessing this property. Similarly, we should not necessarily be looking for the DCN to support the communication patterns mentioned earlier, but for sub-networks within the DCN to do so.

These simple observations lead us to what we feel is a fundamental basic principle of DCN design in relation to the usage of datacenters to support clouds: *a DCN should be viewed as a topology within which other topologies (including itself) have multifarious embeddings and not as an indivisible entity that necessarily possesses specific topological properties in its own right.*

To some extent, this perspective is not new and somewhat obvious (we have already heard how the general embedding problem for DCNs has been shown to be computationally intractable, thus provoking a search for heuristic solutions often based upon an iterative search through sub-networks). However, the key point is that, as yet, datacenter usage has not really fed explicitly into DCN design. The drivers for embedding from general interconnection networks remain but the necessity for virtualization raises their importance.

Finally, although we take a compositional view of DCNs, we cannot ignore some topological aspects of the DCN as a whole that arise because of usage-related issues. For example, we saw earlier that both SecondNet and Oktopus have a centralized manager to collect, process, and disseminate information. In order that this manager can do this, a spanning tree (at least) needs to be identified within the DCN. Thus, we cannot wholly ignore the overall topology of the DCN (we shall revisit this remark later when we discuss recursively defined DCNs).

## 4 Symmetry

Our study of virtualization and communication patterns has led to our fundamental basic principle, as laid out above, but the problem remains as to how we might ascertain or measure how well a DCN design adheres to this principle. Our principle is concerned with the internal structure of a topology in relation to the whole and thus is all about symmetry. In this section, we compare and contrast established notions of symmetry within distributed-memory multiprocessors and networks-on-chips with what might be required of symmetry within a DCN (which we interpret broadly, recall).

### 4.1 Symmetry in distributed-memory multiprocessors

‘Symmetry’ in interconnection networks is regarded as a good thing, and in distributed-memory multiprocessors it means node- and link-symmetry (also called node- and link-transitivity, respectively).

**Definition 2** A graph  $G = (V, E)$  is *node-symmetric* if given any  $u, v \in V$ , there exists an automorphism  $\rho$  of  $G$  so that  $\rho(u) = v$ , where an *automorphism*  $\rho$  of  $G$  is a bijection from  $V$  to  $V$  such that if  $(u, v) \in E$  then  $(\rho(u), \rho(v)) \in E$ . A graph  $G = (V, E)$  is *link-symmetric* if given any  $(u, v), (u', v') \in E$ , there is an automorphism  $\rho$  of  $G$  such that either  $\rho(u) = u'$  and  $\rho(v) = v'$  or  $\rho(u) = v'$  and  $\rho(v) = u'$ .

Node-symmetry means, amongst other things, that each individual processor can be supplied with the same program, routing is vastly simplified (as is the implementation of communication patterns such as many-to-many), and network analysis is

easier [15, 35]. Implicit is the assumption that every processor is working on the same computational task. Furthermore, algebraic characterizations of networks as *Cayley graphs*, so that node-symmetry immediately follows, can yield an even more beneficial environment (see, for example, [2, 35, 43, 67]). On the other hand, link-symmetry can yield well balanced traffic loads [15]. Of course, such symmetry is only part of the story, as traffic patterns, routing algorithms, fault tolerance, packaging constraints, and so on, all have their part to play (and likewise do so in DCNs). Nevertheless, (traditionally defined) symmetric interconnection networks have proved their worth in practice.

## 4.2 Symmetry in DCNs

The term ‘symmetry’ is widely used in the context of DCNs too. However, its meaning in the DCN context has yet to be succinctly defined as in the distributed-memory multiprocessor context. Unlike distributed-memory multiprocessors, datacenters are certainly not ‘single-task machines’ in that they generally simultaneously undertake a whole range of independent computational activities, under the auspices of different users, and they do this in a flexible fashion (from the perspective of the datacenter owner). There are multiple tasks operating under different routing algorithms and generating different traffic loads and communication patterns. Also, DCNs consist of server-nodes and switch-nodes, with the different types of nodes playing very different roles. Indeed, there is an additional dimension to this heterogeneous nature: datacenter usage is geared much more towards the users than is, for example, a distributed-memory multiprocessor computation. Consequently, datacenters need to have a user-facing capacity and some servers or switches within the datacenter need to handle incoming and outgoing user-oriented communication.

All this might imply that the notions of symmetry from Definition 2 are seemingly irrelevant. However, whilst virtualization leads to our fundamental basic principle, it also demands that there exists some form of centralized control within a datacenter so that locations for virtual machines and virtual datacenters can be chosen and managed. This means that there must also be some capacity for the centralized collection of data such as existing link and node loads. Thus, a DCN needs to be able to support (a limited amount of) global communication, for example, via spanning trees or other spanning networks (see, for example, [4, 30, 47] for more on this).

We propose here that symmetry with regard to server-centric DCNs should be with respect to the server-nodes, which is where all the intelligence lies, with the switch-nodes simply being regarded as providing conduits between server-nodes. For example, consider routing a message within a DCN: it is a server-node that initiates this message and the destination of the message is a server-node too. Key to our consideration of symmetry within DCNs will be our fundamental basic principle of datacenter design from Sect. 3.3, and we will move towards formulating more relaxed notions of symmetry, than those in Definition 2, that still reflect beneficial practical datacenter properties. These properties include not only facilitating routing and analysis, as in the case of distributed-memory multiprocessors, but also the identification

of sub-networks, the embedding of (guest) topologies, and the provision of support for implementing communication patterns within the resulting sub-networks.

So, to summarize, symmetry within DCNs should:

- facilitate multiple embeddings of chosen topologies, e.g. virtual clusters and over-subscribed virtual clusters (from [4]), point-to-point pairs (from [30]), and possibly other topologies such as hypercubes, de Bruijn networks, and even other DCNs, so that these embeddings can be efficiently identified (via an algebraic description of the DCN that is amenable to combinatorial and algorithmic manipulation); and
- support the implementation of specific communication patterns within specific sub-networks as well as global communication across the DCN (so that data collection and dissemination can be undertaken).

As is ever the case with interconnection networks, there will be no silver bullet; that is, there will not exist DCNs that are optimal as regards all aspects of symmetry. What is more, symmetry is but one aspect of interconnection network design and there are many others to consider, some of which might be in conflict with beneficial aspects of symmetry.

## 5 Illustrative DCNs

We will soon look at specific aspects of DCN symmetry that support virtualization and the implementation of communication patterns, and we will illustrate these aspects using the DCN HCN [32]. We choose HCN as our illustrative vehicle due to convenience (the DCN HCN is easy to define and to visualize) and because it possesses an intimate relationship with an existing interconnection network; not necessarily because it possesses all of the symmetric properties we highlight below and believe important. As we stated above, symmetry is claimed for HCN in [32] without actually being defined; however, we will see that this DCN is indeed ‘symmetric’ in certain senses relevant to datacenter design. We could equally well have used alternative server-centric DCNs as illustrative vehicles.

The DCN HCN forms the first layer of the two-layer DCN BCN; more precisely, lots of copies of HCN do. However, the constructions of the two layers are distinct in the following sense: all switch-nodes have adjacent server-nodes in the form of master-nodes and slave-nodes (so called in [32]): at the first layer, where a DCN HCN is constructed, the construction uses only the master-nodes, and at the second layer, so as to obtain the DCN BCN, the construction uses only the slave-nodes of different copies of HCN. Consequently, the switch-nodes provide the ‘points of contact’ between two algebraically distinct DCNs, with the second layer construction of the DCN BCN ‘overlaid’ on the first-layer construction of the DCN HCN. (In fact, one could reverse the order of construction of BCN, by building the second layer first, and still obtain the same resulting DCN.) As we do not work with BCN in what follows, we simply refer the reader to [32] for full details of the construction of BCN.

So that we might focus on the DCN HCN, let us simply remove the slave-nodes from all switch-nodes (so, we are just considering the first-layer construction highlighted above). Also, with respect to our intention, laid out above, to focus symmetry



on server-nodes and to regard switch-nodes as providing conduits between server-nodes, we can abstract a switch-node joining  $n$  server-nodes, say, as a clique of links involving these  $n$  server-nodes; we sometimes refer to this abstraction of the DCN as its *clique-abstraction*. When one does this for HCN, what one actually obtains is the interconnection network known as a *WK-recursive network* which originated in [55] and which has since been studied in some detail within the context of distributed-memory multiprocessors and networks-on-chips (this observation, linking HCN and the WK-recursive network, was originally made in [23]).

**Definition 3** The DCN  $HCN(n, h)$ , with its slave-nodes removed (see [32]) and with every switch-node replaced by a ‘clique of links’ joining the server-nodes adjacent to the switch-node, has node set  $\{1, 2, \dots, n\}^{h+1}$ . There are links of the form

$$((i_h, i_{h-1}, \dots, i_2, i_1, x), (i_h, i_{h-1}, \dots, i_2, i_1, y)),$$

whenever  $x \neq y$ , and also links of the form

$$\begin{aligned} &((i_h, i_{h-1}, \dots, i_{j+1}, i_j, i'_j, \dots, j\text{times} \dots, i'_j) \\ &(i_h, i_{h-1}, \dots, i_{j+1}, i'_j, i_j, \dots, j\text{times} \dots, i_j)), \end{aligned}$$

where  $j \in \{1, 2, \dots, h\}$  and  $i_j \neq i'_j$ . The parameter  $n$  details the degree of any switch-node and the parameter  $h$  the level or depth of the recursive construction. When considering  $HCN(n, h)$  as a WK-recursive network, we refer to the parameter  $n$  (that is, the size of the base cliques) as the *amplitude*.

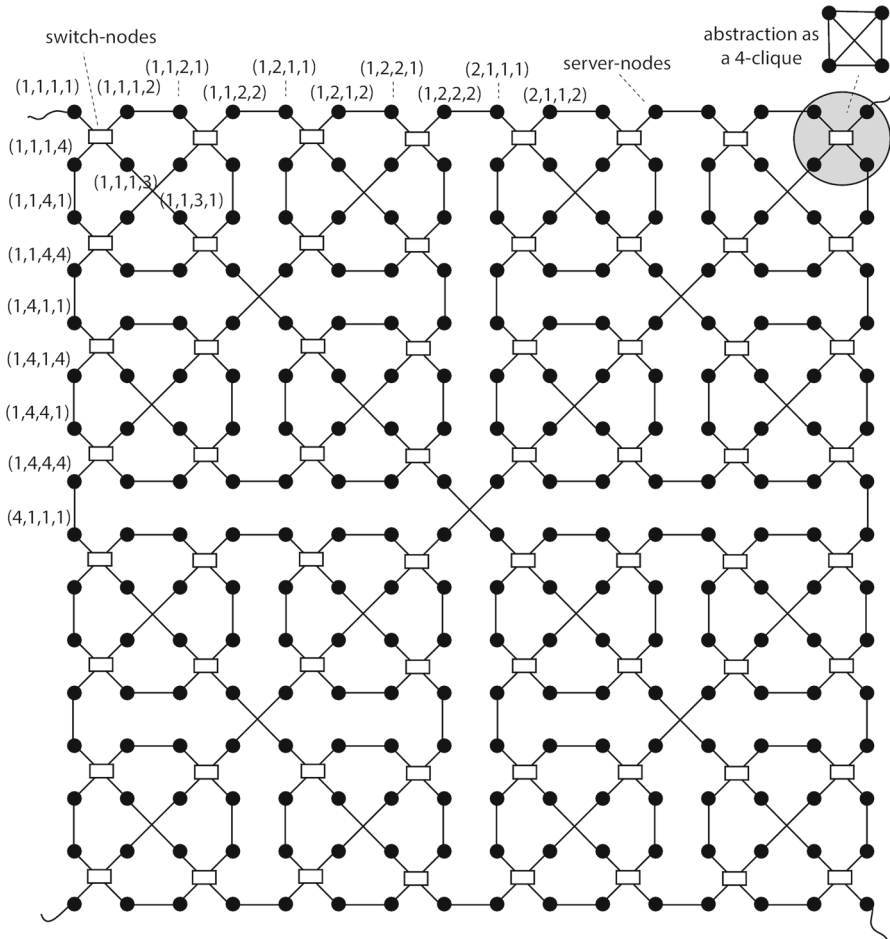
The DCN  $HCN(4, 3)$  can be visualized as shown in Fig. 2 (where we also show how a switch can be abstracted as a 4-clique). Note how there are potential additional links involving the ‘corner’ nodes that can be used to construct  $HCN(4, 4)$  (there is a hint given in Fig. 2 as regards the naming scheme; see [32] for more details).

## 6 Aspects of symmetry in DCNs

In this section, we argue as to why specific symmetry properties of DCNs are beneficial as regards various aspects of datacenter usage encompassing virtualization and the implementation of communication patterns (primarily many-to-many). We describe these symmetry properties and illustrate them (when we can) in the DCN HCN (and briefly in other DCNs). Our aim is to justify why these symmetry properties should be important parameters as regards future (server-centric) DCN design. In the next section, we outline how some of these symmetry properties might be utilized in the design of new server-centric virtualization methodologies.

### 6.1 Recursively defined DCNs

Let us begin by looking at the DCN  $HCN(n, h)$  where there is obvious recursive symmetry. The sub-networks obtained by fixing the first component of all node



**Fig. 2** A visualization of  $HCN(4, 3)$

names form  $n$  copies of  $HCN(n, h - 1)$ . With regard to Fig. 2, the corresponding 4 copies of  $HCN(4, 2)$  are related via the automorphism obtained by a rotation clockwise through  $90^\circ$ . Furthermore, there are numerous other copies of  $HCN(n, h - 1)$  within  $HCN(n, h)$ . Again with reference to Fig. 2, in  $HCN(4, 3)$  there are copies of  $HCN(4, 2)$  identified by their ‘top-left corner-nodes’  $(1, 2, 1, 1)$ ,  $(1, 4, 1, 1)$ ,  $(1, 3, 1, 1)$ ,  $(2, 4, 1, 1)$  and  $(3, 1, 1, 1)$ . Similarly, by fixing the first two components of all node names, we obtain  $n^2$  copies of  $HCN(n, h - 2)$  within  $HCN(n, h)$ . In fact, the DCN  $HCN$  is an example of what are commonly called recursively defined interconnection networks.

**Definition 4** A family  $\{X(h): h = 0, 1, \dots\}$  of interconnection networks is *recursively defined* if the network  $X(h)$ , where  $h > 0$ , is the disjoint union of copies of  $X(h - 1)$  with the inclusion of additional nodes and/or links interconnecting the disjoint copies.

We need to adapt Definition 4 to DCNs.

**Definition 5** A family  $\{X(h): h = 0, 1, \dots\}$  of DCNs is *recursively defined* if the network  $X(h)$ , where  $h > 0$ , is the disjoint union of copies of  $X(h - 1)$  with the inclusion of additional server-nodes, switch-nodes and/or links interconnecting the different copies.

The focus as regards recursively defined DCNs is that the constituent ‘sub-DCNs’ should be ‘glued together’ using (limited) additional resources, so that these constituent sub-DCNs remain available within the larger DCN.

Consider HCN again. It is clearly the case that HCN is recursively defined: here, each  $HCN(n, h)$  contains disjoint, constituent  $HCN(n, h - 1)$ s, with the additional resources used consisting of a relatively small number of links. An analogous statement can be made as regards DCell and FiConn. The DCN BCube is also recursively defined, but now the additional resources involve links and switch-nodes. The DCN SWCube, formed from a generalized hypercube by regarding the nodes as switch-nodes and sub-dividing each edge with a server-node, is also recursively defined, but the additional resources come in the form of additional server-nodes, switch-nodes, and links (SWKautz and SWdBuijn are similarly constructed from Kautz digraphs and de Bruijn digraphs, respectively). However, BCN, DPillar, SWKautz, and SWdBuijn are not recursively defined, for lower-level sub-DCNs do not exist as constituent copies within a larger DCN; these networks do have a ‘recursive flavour’, but do not adhere to Definition 5, which, as we shall argue now, is what is required in order to best support virtualization and communication patterns.

### 6.1.1 The automatic provision of virtual datacenters in clouds

The property of a DCN being recursively defined is not just of idle curiosity (and in keeping with our fundamental basic principle from Sect. 3.3); it is an important aspect of DCN design. Recall that both SecondNet and Oktopus operate so that sub-networks within the underlying DCN are of fundamental importance: with SecondNet, sub-networks (of differing sizes) are used as a search space within which embeddings might be obtained (minimal consideration is given in [30] as to how sub-networks might be defined for consideration within SecondNet), and with Oktopus, tree-like topologies of increasing depth within the host DCN are used likewise as regards the embedding of virtual clusters and oversubscribed virtual clusters (Oktopus is inextricably tied to operation within tree-like DCNs). Having recursively defined DCNs yields two significant advantages: first, there is a ready-made and uniformly structured notion of sub-networks of different sizes within the DCN, namely the lower-level recursive sub-DCNs; second, the recursive nature of the decompositions means that only one embedding algorithm is required (parameterized by the depth of recursion). Also, as we’ll see in a moment, recursively defined interconnection networks lend themselves to the recursive construction of embedded (spanning) trees and other (tree-like) topologies. Consequently, having a recursively defined DCN facilitates the implementation of virtualization methodologies such as SecondNet and Oktopus: it makes uniform both the search for sub-networks of server-nodes and embedding within sub-DCNs.

Of course, recursive-definability also aids other tasks within DCNs such as routing (we expand upon this later when we discuss the connectivity of recursive-definability).

Allied to a DCN being recursively defined is having an algebraic description of the recursive definition that makes the search for (recursive) sub-structures and the utilization of these sub-structures (with respect to embedding, routing, or whatever) efficient. Each of our DCNs HCN, BCN, BCube, DCell, FiConn, DPillar, SWCube, SWKautz, and SWdBruijn has a concise algebraic description. However, there is still much to do in terms of combinatorially investigating these DCNs. For example: we do not even have a closed form for the number of server-nodes in FiConn; we do not have a shortest-path routing algorithm for DCell, FiConn, and BCN; and we have only recently determined the diameter of DPillar [21] and a shortest-path routing algorithm for DPillar [21] and HCN [23]. Of course, hardly anything is known as regards automorphisms of these DCNs that would prove practically useful as regards virtualization (by yielding mappings between embedded sub-structures).

### 6.1.2 The connectivity of recursive-definability

What can be important as regards recursively defined interconnection networks is the pattern of interconnection as one builds higher-level networks out of lower-level networks. In what follows, when we say that a network is recursively defined, we assume that there are canonical copies of lower-level networks around which the recursive definition is phrased.

**Definition 6** A recursively defined family  $\{X(h): h = 0, 1, \dots\}$  of interconnection networks is *completely connected* (see, for example, [9]) if  $X(0)$  is connected and the following holds:

- for each pair of distinct (canonical) copies  $Y$  and  $Z$  of  $X(h - 1)$  in  $X(h)$ , there is a path from a node of  $Y$  to a node of  $Z$  using only the nodes and links that we added to build  $X(h)$  from the copies of  $X(h - 1)$ .

Again, we have to adapt Definition 6 for DCNs.

**Definition 7** A recursively defined family  $\{X(h): h = 0, 1, \dots\}$  of DCNs is *completely connected* if  $X(0)$  is connected and the following holds:

- for each pair of distinct (canonical) copies  $Y$  and  $Z$  of  $X(h - 1)$  in  $X(h)$ , there is a path, which we call a *linkage*, from a server-node of  $Y$  to a server-node of  $Z$  using only the server-nodes, switch-nodes, and links that we added to build  $X(h)$  from the copies of  $X(h - 1)$ .

However, it is possible that linkages share server-nodes, switch-nodes, or links, and consequently, the simultaneous use of these paths might incur congestion. If we can find linkages that are mutually pairwise internally disjoint, then we say that our recursively defined DCN is *strongly completely connected*. Each of the recursively defined DCNs HCN, BCube, DCell, FiConn, and SWCube is strongly completely connected.

Whilst it is clear that (strongly) completely connected recursively defined DCNs provide additional flexibility as regards embedding virtual datacenters, they also facilitate support for routing as we now demonstrate. Let  $\{X(h): h = 0, 1, \dots\}$  be any

completely connected recursively defined family of DCNs. Given any two copies of  $X(h-1)$  within  $X(h)$ , there is at least one linkage incident with a node in both copies. There is a canonical routing algorithm for the family  $X$  as follows:

- if the source node  $s$  and the destination node  $t$  lie in the same copy of  $X(h-1)$  within  $X(h)$ , then route the message from  $s$  to  $t$  recursively
- if the source node  $s$  and the destination node  $t$  lie in different copies of  $X(h-1)$  within  $X(h)$ , then:
  - find a linkage joining the two copies of  $X(h-1)$
  - route the message recursively from  $s$  to the node in the copy of  $X(h-1)$  containing  $s$  that is incident with the linkage
  - route the message over the linkage
  - route the message recursively to  $t$  from the node in the copy of  $X(h-1)$  containing  $t$  that is incident with the linkage.

Clearly there are different routing algorithms available when there is more than one linkage joining copy of  $X(h-1)$  in  $X(h)$ ; also, the more the linkages that are available, the more the scope there is for improving throughput and fault tolerance. It should be noted that  $\text{HCN}(n, h)$  is such that there is only 1 linkage between any two copies of  $\text{HCN}(n, h-1)$  within  $\text{HCN}(n, h)$ , with analogous statements as regards the DCNs DCell and FiConn. However, there are numerous linkages joining two corresponding sub-DCNs in SWCube and BCube.

We should remark that the above canonical routing algorithm need not be such that the paths obtained are the shortest possible. For example, as regards WK-recursive networks, and so HCN, the canonical routing algorithm resulting from the description above is that from [10, Sect. 3.1] where it is noted that sometimes the paths obtained are not shortest paths. An improved algorithm is presented in [10, Sect. 3.2]. The degree of improvement was experimentally validated in [23] where a practical analysis of the resulting routing algorithms for the DCNs HCN and BCN is undertaken. Also, in [19], the completely connected nature of FiConn and DCell was utilized to develop improved routing algorithms, beyond those presented in [31] and [44], by using the numerous alternative ('proxy') routes available. However, irrespective of whether canonical algorithms in some DCN can be improved or not, having completely connected recursively defined DCNs means that these canonical algorithms are readily available (assuming that the algebraic description of the DCN enables a straightforward implementation, which is the case for HCN, BCube, DCell, FiConn, and SWCube).

Not only does a completely connected recursively defined DCN support virtualization and routing algorithms, but it also supports the generation of spanning trees and the evolution of spanning trees in the presence of faults; recall that both SecondNet and Oktopus need to establish spanning trees (for use by a centralized manager) and it is explicitly stated in [30] that SecondNet has the capacity to evolve this spanning tree in the presence of faults. The generation of a spanning tree is by an obvious recursive algorithm and the fact that the DCN is completely connected obviously provides some flexibility as regards this construction (particularly in the presence of a limited number of faults).

### 6.1.3 The flexibility of the recursive decomposition

Many standard interconnection networks, like the families of hypercubes and  $k$ -ary  $n$ -cubes, are recursively defined; moreover, they have additional flexibility as regards their recursive structure. Consider the  $n$ -dimensional hypercube  $Q_n$  for example. The interconnection network  $Q_n$  is constructed from two copies of  $Q_{n-1}$  so that adjoining links are added between every node of one copy of  $Q_{n-1}$  and its counterpart in the other copy. However, we can partition  $Q_n$  in this way by choosing to partition over any one of  $n$  dimensions. Such flexibility to the recursive decomposition clearly translates into flexibility as regards embedding where there is more potential to find sub-networks within which to embed or so as to avoid faults. The DCN SWCube shares this additional flexibility as regards partitioning. On the other hand, the DCNs HCN, BCube, DCell, and FiConn only have one way to undertake a recursive partition.

As we saw earlier,  $\text{HCN}(n, h)$  does possess various embedded copies of  $\text{HCN}(n, h-1)$  (even though there is only one recursive decomposition). Any recursive embedding algorithm will clearly benefit from having access to a variety of sub-networks. As regards recursively defined interconnection networks, the existence of a sub-network  $X(h-1)$  within a host network  $X(h)$  has been considered via studies on *reliability*: each node is apportioned some failure probability and an analysis of  $X(h)$  is undertaken as to the likelihood of there existing a healthy copy of  $X(h-1)$  within  $X(h)$ . This model for reliability was first proposed in [8] (see, for example, [50] for some more recent developments). Of course, an analogous analysis of reliability in DCNs would be directly relevant to embedding within recursively defined DCNs, but, as far as we are aware, no such reliability analysis exists (the only consideration of notions of reliability in DCNs that we know of can be found in [13]).

In summary, the following aspects of symmetry are important so as to support virtualization within DCNs.

- A DCN should be recursively defined so that the recursive structure is completely connected with a choice of linkages between two recursive components. In addition, there should exist numerous copies of sub-DCNs with a host DCN. The recursive definition of the DCN should be algebraically concise.

## 6.2 Support for communication patterns

As we have seen, DCNs need to support various communication patterns, such as many-to-one, one-to-many, many-to-many, and so on, within sub-networks of the DCN. Note that if a DCN is recursively defined, then this essentially means that the DCN itself needs to support such communication patterns (this qualifies the remark we made right at the end of Sect. 3.3). The most common method by which these communication patterns are supported is by using spanning trees (see, for example, [4, 12, 29–31, 46, 47, 64]). Indeed, BCube [29] and Camdoop [12] utilize multiple edge-disjoint spanning trees.

The analysis of a DCN as regards its capacity to support many-to-many broadcasts is usually undertaken by measuring the aggregate bottleneck throughput. The *aggregate bottleneck throughput* was established in [29] and is defined as the number of flows

multiplied by the throughput of the bottleneck flow, where the bottleneck flow is the flow that receives the smallest throughput (by a *flow* we mean a path from a source to a sink complete with a data load; flows are generally used to refer to the transportation of significant amounts of data where the lifespan of the reserved path is non-trivial). The measurement of the aggregate bottleneck throughput (see, for example, [29,33,44]) is not usually undertaken with respect to a specific routing algorithm; rather, it is assumed that all flows traverse a shortest path between the two corresponding server-nodes. Consequently, the calculation of the bottleneck flow is not always reflective of the variations in load caused by employing a specific routing algorithm. In short, the underlying DCN topology and routing algorithms are generally not taken fully into account in an analysis of many-to-many broadcast support in server-centric DCNs.

There is another important point to note with regard to many-to-many broadcasts in relation to supporting MapReduce: it is often the case that the key–value pairs generated by a map task do not constitute a significant amount of data, so that one can treat this data *en masse*. This yields an alternative method by which a many-to-many broadcast relating to MapReduce can be undertaken: if the reducers are interconnected in the form of a closed path and these reducers are also the mappers, then the different batches of key–value pairs generated by each mapper can be ‘daisy-chained’ around the closed path with each reducer pulling out the key–value pairs from the packet (or small number of packets) that are relevant to it (by ‘daisy chain’ we mean a mapper receives data from the previous mapper on the closed path, removes data intended for itself, and passes on the remaining data to the next node on the closed path). The daisy-chaining implementation might also help to smooth out traffic spikes. Obviously it is preferable (so as to avoid undue congestion) that no server-node nor link appears more than once on this closed path.

For example, consider the DCN HCN(4, 3), as depicted in Fig. 2. If we abstract switch-nodes as 4-cliques of server-nodes (that is, we are working with the clique-abstraction of HCN), then it is not difficult to derive a Hamiltonian cycle; the same applies to HCN(4, 2) and HCN(4, 1). Whilst a cycle in the abstracted WK-recursive network does not necessarily translate to a cycle in HCN, it does translate to a closed path containing the corresponding server-nodes, but where a switch-node might appear more than once (this is because a server-node-to-server-node path through a switch-node in HCN is abstracted as a link in the WK-recursive network). However, this causes no additional congestion or other problems when it comes to ‘daisy-chaining’ data around this closed path in the DCN HCN (as switches in datacenters are non-blocking). Consequently, sometimes when we say that there is a cycle in some DCN, what we mean is that there is a closed path corresponding to a cycle in the clique-abstraction of the DCN. Similarly, when we say that a DCN is Hamiltonian, what we mean is that there is a Hamiltonian cycle in the clique-abstraction of the DCN.

In order that we might utilize cycles within a DCN so as to facilitate many-to-many broadcasts relating to MapReduce, it is preferable that we have numerous cycles to choose from and that these cycles are widespread within the DCN (recall that virtualization dictates that sub-networks within our DCN might be used by some tenant to undertake a MapReduce and that we have already examined in detail the need for our DCNs to be amenable and flexible as regards simultaneously accommodating numer-

ous virtual datacenters). A related concept within general interconnection networks intuitively reflects the existence of cycles.

**Definition 8** An interconnection network  $X$  on  $n$  nodes is *pancyclic* if there is a cycle of every length from 3 to  $n$  in  $X$ , and *node-pancyclic* (resp. *link-pancyclic*) if every node (resp. link) lies on a cycle in  $X$  of every length from 3 to  $n$ .

Node- and link-pancyclicity are clearly properties relating to symmetry as they reflect the existence of cycles specific to any particular node or link of the interconnection network. There are a number of variations in the concept of pancyclicity in the literature (see, for example, [37]).

Consider the DCN HCN abstracted as a WK-recursive network by replacing switch-nodes with cliques. The WK-recursive network is pancyclic, so long as the amplitude is at least 5 [24]. In fact, it was further proved in [26] that if the amplitude is at least 6, then the WK-recursive network is node-pancyclic, and that whilst the WK-recursive network of amplitude at least 7 is not link-pancyclic, there exists an  $m$  (depending on the amplitude) for which given any link, there is a cycle of any length at least  $m$  passing through that link. Hence, there is considerable scope for finding cycles in the DCN HCN.

In the absence of pancyclicity and in the situation where our DCN is recursively definable, the existence of spanning cycles within each recursive copy would be beneficial (in the above context). Of course, given the recursive-definability, this amounts to the DCN being Hamiltonian (which is trivially the case for WK-recursive networks, given the above). Whilst a Hamiltonian (recursively defined) DCN facilitates many-to-many broadcasts, stronger properties can improve things even further. For example, it is proved in [39] that a WK-recursive network of amplitude  $2n + 1$  has  $n$  link-disjoint Hamiltonian cycles from which an almost optimal all-to-all broadcast can be developed (here, optimality is with respect to an all-port model of computation). The following property provides additional flexibility with regard to finding Hamiltonian cycles.

**Definition 9** An interconnection network  $X$  is *Hamiltonian-connected* if there is a Hamiltonian path joining any two distinct nodes.

Of course, when we say that a DCN is Hamiltonian-connected, we mean that its clique-abstraction is.

The DCNs in this paper have not been extensively studied as to whether they have properties relating to pancyclicity and Hamiltonicity. However, there are a few results known and we can sometimes use the relationship of a DCN with an existing interconnection network in order to use existing results. It was proved in [25] that any WK-recursive network of amplitude at least 4 is Hamiltonian-connected; consequently, HCN is Hamiltonian-connected (when all switch-nodes have degree at least 4). It was shown in [57] that (apart from a very small number of cases) DCell is Hamiltonian-connected and remains Hamiltonian-connected even in the presence of (a limited number of) faults. The fact that DCell is strongly completely connected means that it has numerous useful cycles embedded, with the Hamiltonian connectedness adding to the flexibility of finding these cycles (it should be added that an algorithm to find a Hamiltonian path in DCell is given in [57]). As was remarked in [57], the constructions



used there do not apply to FiConn and do not yield pancyclicity results for DCell. The clique-abstraction of the DCN BCube is the generalized hypercube. It has been proved in [38] that the generalized hypercube is Hamiltonian-connected and pancyclic; consequently, BCube has these properties too. The clique-abstraction of the DCN DPillar contains a wrapped butterfly network as a spanning subgraph; consequently, as a wrapped butterfly network is Hamiltonian [58], so is DPillar. As we have mentioned, SWCube is derived from the generalized hypercube by replacing each node with a switch-node and sub-dividing every link with a server-node. However, it is not immediately apparent as to whether Hamiltonicity and pancyclicity results for the generalized hypercube translate into analogous results for SWCube; essentially, the construction of the DCN SWCube from a generalized hypercube, as described above, corresponds to taking the *line graph* of a generalized hypercube.

Other aspects of symmetry can support different communication patterns. We have heard how spanning trees feature widely in supporting communication. Suppose that we take the clique-abstraction of a DCN and that this graph is node-symmetric. So, given a source and a target server-node, there is an automorphism mapping the source to the target. Consequently, we can choose to 're-root' any spanning tree to any chosen server-node by taking its image under an appropriate automorphism; this gives us added flexibility as to how we utilize spanning trees. Algebraic aspects of interconnection networks should not be underestimated. For example, algebraic constructions are used in [36] to develop an all-to-all broadcast algorithm for Cayley graphs where the resulting paths are all shortest paths and where there is a uniform load on nodes.

Some DCN constructions, such as that for FiConn, are not homogeneous; with FiConn, some server-nodes have degree 1 and some degree 2. Here, traditional node-symmetry is not the concept that is relevant to us. What is important is the existence of automorphisms of (the clique-abstraction of) our DCN so that the number of orbits is as small as possible, where an *orbit* in this context is a set of server-nodes each of which can be mapped to any other in the set via an automorphism of the DCN. The server-nodes of each orbit can be handled similarly; for example, a spanning tree rooted at one server-node can be algebraically transformed into a spanning tree rooted at any other server-node in the same orbit. For example, in HCN(4, 1) the 'corner-nodes' clearly form an orbit and it can easily be shown that there is an automorphism from any non-corner server-node to any other non-corner server-node; thus, the server-nodes are partitioned into 2 orbits. The study of the automorphisms of server-centric DCNs has hitherto not been undertaken; indeed, this discussion is the first mention of the relevance of automorphisms within DCN design.

Finally, let us return to the computation of the aggregate bottleneck throughput that we highlighted earlier. As we noted, this computation is generally not undertaken with respect to specific routing algorithms. However, if our DCN is recursively defined then, as we explained earlier, there are obvious methods to obtain canonical routing algorithms. Consequently, this framework lends itself to a more accurate analysis of aggregate bottleneck throughput.

In summary, the following aspects of symmetry are important so as to support many-to-many broadcasts (relating to MapReduce) as well as other communication patterns and routing in DCNs.

- A DCN should contain numerous cycles of various lengths, a property that is reflected in the DCN being pancyclic (or some variation on this theme). In the absence of pancyclicity, a recursively defined DCN should be Hamiltonian.
- The use of spanning trees to support communication patterns is best undertaken within DCNs for which the number of orbits of server-nodes (under automorphisms) is small.

### 6.3 Hierarchical DCNs

We end this section with a brief consideration of hierarchical DCNs, motivated by the construction of the DCNs BCN from (or on top of) the DCNs HCN in [32]; a DCN or interconnection network is *hierarchical* if it is constructed from a ‘fusion’ of different methodologies, e.g. by superimposing one network on another or identifying nodes of two distinct networks. It is worthwhile commenting on the extension of HCN to BCN and how this relates to symmetry. First, we explain how to define the DCN BCN from the DCN HCN.

Suppose that we have a graph  $G$  on  $n$  nodes. We can take  $n + 1$  copies of  $G$ , say  $G_0, G_1, \dots, G_n$ , and add  $\frac{n(n+1)}{2}$  additional links so that for every distinct  $i, j \in \{0, 1, \dots, n\}$ , there is exactly one link joining a node in  $G_i$  to a node in  $G_j$  and every node is incident with exactly one new link (this can be done in a number of ways). In fact, this is essentially the iterative construction used to build the DCN DCell [31].

However, rather than do this with  $\text{HCN}(n, h)$  replacing  $G$ , above, we can alternatively attach to every switch-node of  $\text{HCN}(n, h)$   $m$  server-nodes called slave-nodes (recall that we ignored these slave-nodes earlier when we worked with  $\text{HCN}(n, h)$ ). So, there are  $mn^h$  slave-nodes adjacent to switch-nodes in  $\text{HCN}(n, h)$ . We now take  $mn^h + 1$  copies of  $\text{HCN}(n, h)$  and undertake the above DCell construction with respect to the slave-nodes (as to which links we introduce is clearly defined in [32], although we can actually introduce these links in a variety of ways; see [31]).

We can extend this construction. We could consider  $\text{HCN}(n, h)$  as consisting of, for example, (the canonical)  $n^2$  copies of  $\text{HCN}(n, h - 2)$ ; each of these copies has  $mn^{h-2}$  slave-nodes. We might now take  $mn^{h-2} + 1$  copies of  $\text{HCN}(n, h)$  and join corresponding copies of  $\text{HCN}(n, h - 2)$  according to the DCell construction above. Full details can be found in [32] (again, as to which links we introduce is clearly defined in [32]). What is sufficient for us is that the DCN BCN is formed from disjoint copies of  $\text{HCN}(n, h)$  by overlaying the DCell construction from [31] (and there are a number of ways to do this).

The question is: how does the DCN BCN conform to our notions of symmetry developed so far? In a sense, the formation of any BCN (no matter which construction is adopted) is a recursive construction, albeit of a different nature to the one used to build HCN. It introduces many more copies of  $\text{HCN}(n, i)$ , where  $0 \leq i \leq n$ , and since we have argued that the DCN HCN is symmetric, in many of the senses we have discussed, this can only be a good thing. A negative aspect of the BCN construction is that it ‘seals’ the DCN, as once one has applied the BCN construction once, one can go no further. The hierarchical DCN BCN is a fusion of two distinct constructions. However, we have yet to fully analyse the DCN DCell in terms of (DCN notions of)

symmetry, and, of course, we have yet to fully consider the symmetric interactions of the DCN HCN and the overlaying of the DCell construction within BCN. We leave these topics for another time and close by remarking that the concept of overlay constructs, as illustrated by the DCN BCN, is an interesting and as yet undeveloped region of future DCN design.

## 7 A new virtualization methodology

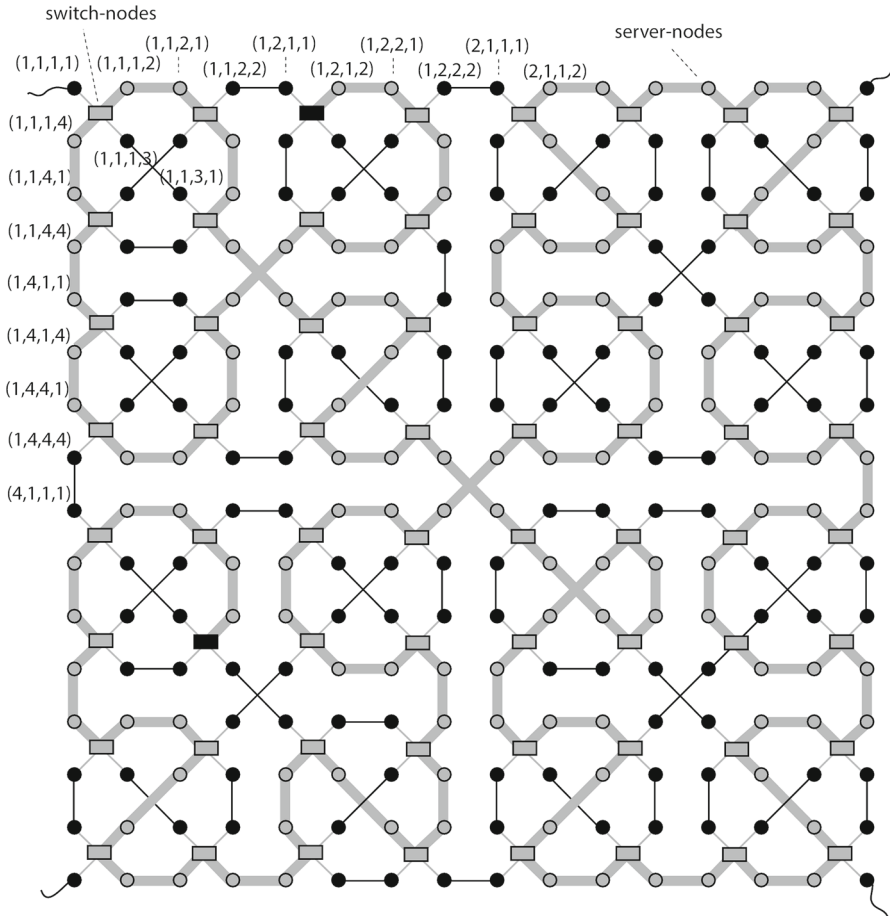
Let us now turn to a new embedding methodology for virtualization. We look again at aspects of symmetry in DCNs that aid virtualization but we do this in tandem with a new methodology to enable virtualization in server-centric DCNs. In so far as we are aware, this is the first real attempt to, first, consider virtualization in (non-tree-based) server-centric DCNs, and, second, explicitly use the underlying DCN topology when embedding virtual DCNs (so addressing a specific direction for future research as proposed in [5]). However, as we explain later, our methodology is not as topology specific as one might think as it exploits properties of symmetry inherent within many server-centric DCNs, and also highlights properties we would wish of new DCNs in order that virtualization is better supported.

We begin by highlighting our new methodology and its potential benefits; these benefits are with regard to the practicalities of virtualization. We only describe our new methodology in sufficient detail so that key design concepts can be grasped; the full implementation of our methodology, and the necessary empirical experimentation and analysis, is beyond the scope of this paper and will be undertaken subsequently. Having outlined our methodology, we look at structural topological properties of DCNs that might support this methodology (we provide enough detail as regards our methodology so that the relevance of the underlying aspects of symmetry can be appreciated). Finally, we briefly review existing server-centric DCNs from the perspective of these properties and, consequently, how supportive these DCNs might be to virtualization.

### 7.1 A new virtualization methodology

Our key observation is simple: a (long) path (or cycle) of server-nodes and switch-nodes can be used to ‘stack’ virtual DCNs. For concreteness, we illustrate our ideas by embedding virtual clusters in the DCN HCN.

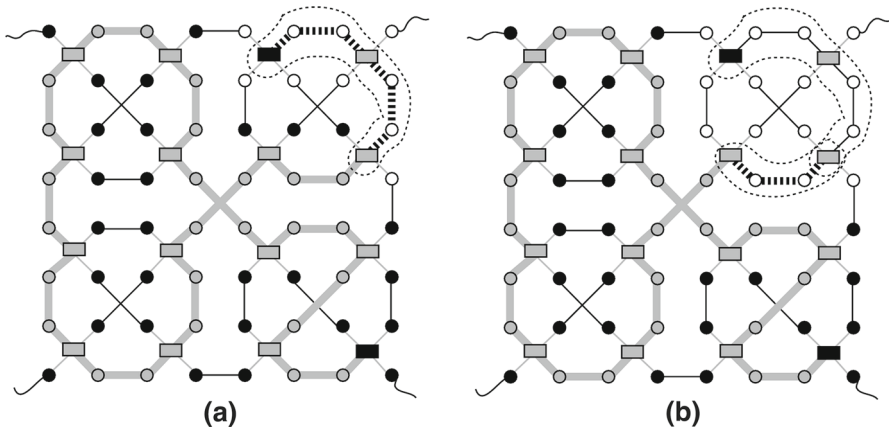
Suppose, for simplicity, that each server-node can support only one virtual machine; that is, the load of any embedding is necessarily 1 (in addition, for simplicity, we ignore other aspects of the host DCN and the virtual cluster such as memory, storage, bandwidth, and so on). Consider the copy of HCN(4, 3) in Fig. 3 and the bold grey path of links (joining the two black switch-nodes whose indices are (1, 2, 1) and (4, 1, 3)). By regarding this bold grey path of links as a virtual switch, we can embed a virtual cluster with 256 server-nodes in HCN(4, 3) (each server-node of the virtual cluster is mapped to a unique server-node of HCN(4, 3); the grey links in Fig. 3 are links connecting black server-nodes to the virtual switch, whereas the grey server-nodes are already within the virtual switch). Of course, by taking sub-paths of this path we



**Fig. 3** An embedding path in  $HCN(4, 3)$

can obtain analogous embeddings of virtual clusters with up to 256 server-nodes (in a similar one-to-one matching of virtual machines to host server-nodes).

A more concrete illustration of how we can embed a virtual cluster consisting of 10 virtual machines along a path within  $HCN(3, 3)$  is visualized in Fig. 4a. Here: the white server-nodes host the 10 virtual machines of the virtual cluster; the sub-network within the dotted lines implements the virtual switch; and the bold dotted black links are the links which bear the brunt of the traffic. Note how the sub-network hosting the virtual cluster is a tree; actually, it is a *caterpillar* (a tree where all nodes are adjacent to nodes on a central path) with nodes that can be server-nodes or switch-nodes. Whereas a virtual machine-to-virtual machine route in the virtual cluster is simply a path from the source virtual machine to the virtual switch and on to the target virtual machine, in the worst case this path maps to a path in the host DCN consisting of 8 links; consequently, depending upon the traffic pattern, there will be overheads, in terms of



**Fig. 4** Virtual cluster embeddings in HCN(3, 3)

latency and congestion, to be borne within the host DCN (but, of course, this is true no matter which methodology one uses to embed).

However, with reference to Fig. 4, the linear nature of our embedded path enables us to ‘stack’ additional virtual cluster embeddings one after the other along this path, in a convenient and easy-to-maintain fashion. For example, suppose that we had an additional virtual cluster to embed where this additional virtual cluster consists of 5 virtual machines. We could stack this virtual cluster on our path by using the 2 unused server-nodes adjacent to one of the switch-nodes involved in the embedding of the first virtual cluster, along with 3 server-nodes adjacent to the next switch-node on the path, as depicted in Fig. 4b. In this way, we can use the structural property of a DCN having a long path or cycle to store and organize the embedding of virtual clusters. We now highlight the potential benefits of our approach.

### 7.1.1 Migration

Virtual clusters come and go within virtualization, and our methodology lends itself to the allocation and migration of virtual cluster embedding. For example, suppose that our first virtual cluster (consisting of 10 virtual machines) terminates. We have a choice of either reusing the freed server-nodes by embedding subsequent virtual clusters or we can migrate existing embeddings by ‘sliding’ them down our stack towards the source. So, for instance, our second virtual cluster (of 5 virtual machines) could be re-embedded using the 4 server-nodes adjacent to the black switch-node (in Fig. 4) together with an adjacent server-node (of course, we still embed within our chosen caterpillar within HCN(3, 3)). Moreover, any other embedded virtual clusters can be ‘slid’ down the stack in exactly the same way. Consequently, we can easily migrate existing embeddings of virtual clusters in order to group together (in terms of locality and according to whatever migration strategy we choose to employ) large numbers of unused server-nodes and switch-nodes so as to provide capacity for future embeddings. Of course, the locality inherent within the path facilitates this migration of

virtual clusters by limiting network traffic generated by the re-embedding. In practice, of course, the migration of virtual cluster embeddings will be determined by a number of factors such as the fragmentation caused by existing embeddings, the (expected) lifespan of existing embeddings, the (expected) arrival of new virtual clusters, and the cost of migration. One can clearly appreciate the ease by which we can cope with migration within our caterpillar when one compares with the analogous situation within, say, a tree.

### 7.1.2 *Locality and global data collection*

Our approach should be compared with the existing (switch-centric) virtualization embedding approaches that we highlighted earlier. The only structural assumptions they make are that servers are organized in racks in a tree-like fashion and that communication is via top-of-rack switches, edge switches, and core switches. The assumption is that the embedding algorithm has complete knowledge of which virtual cluster is embedded where within the host DCN; such is the case for SecondNet, for example, where the DCN collects its information via a spanning tree signalling channel. Of course, there is a cost in that there needs to be a continual collection of data so as to ascertain exactly which virtual clusters have terminated, where resources are free, and whether a migration should be undertaken (though as is noted in [30], this spanning tree is only used for signalling purposes and so the traffic generated within it is light). There is nothing to stop us mirroring SecondNet and generating an analogous signalling spanning tree, where this tree is dynamically constructed as nodes and links become faulty. Alternatively, it could be the case that our path-based methodology provides an alternative signalling mechanism (although we have not pursued this consideration any further).

The nature of existing virtualization algorithms is that ad hoc (relative to the underlying topology) distributions of virtual clusters within the host DCN result; in particular, there is no guarantee of locality (consequently, migration costs might be higher). Our approach guarantees locality and also yields the possibility of improving data collection by limiting migration traffic generated, as we now explain.

A highly localized traffic-limiting approach is to think of data collection being via a ‘window’ that continually moves up and down the caterpillar (or at least the portion of the path within which there are currently virtual clusters embedded) so that when ‘gaps’ are found, embedded virtual clusters are ‘slid’ down the stack so as to fill the gap. Such a ‘sliding window’ approach uses the locality inherent within the path and within the virtual cluster embeddings to limit fragmentation and so facilitate new embeddings. The key point is that such a defragmentation approach could not be employed with embedding methodologies that are tree based or based around ant colony optimization, greedy topology-agnostic heuristics, or linear programming.

### 7.1.3 *Energy efficiency*

Recent attempts to facilitate virtualization have been geared towards energy efficiency and it is appropriate that we highlight potential benefits of our proposed methodology in this light. As we stated earlier, as yet there have been no attempts to tackle virtualization

in a server-centric setting; however, the energy models as regards virtualization and energy efficiency in switch-centric DCNs hold good in server-centric DCNs. The fundamentals concerning energy efficiency are well laid out in [14], for example. Broadly speaking: the energy consumption of a server depends upon the CPU load, with the idle server still consuming a significant fraction of the energy consumed when it is fully loaded; and the energy consumption of a switch is dependent upon the number of ports that are disabled, with a switch with all its ports disabled still consuming a significant fraction of the energy consumed when all ports are enabled. The upshot, from [14], is that (not surprisingly) it makes sense to run CPUs with as high a load as possible and to enable as many switch ports as possible. Note that our methodology supports both intuitive aims: in an unfragmented embedding of virtual clusters along our path, there is at most one switch-node with an enabled port and a disabled port, and there is most one server-node where the CPU is not idle and not fully loaded. Also, it is not difficult to appreciate that our methodology lends itself to powering down unused server-nodes and links (given a virtual cluster embedding scenario).

## 7.2 Symmetry for virtualization

Given our proposed virtualization methodology for server-centric DCNs, we now look at structural topological properties that support the usage of this methodology and so virtualization (on the grounds of simplicity, we continue to embed virtual clusters).

Ideally, we want a path upon which every switch-node lies and is such that every server-node is either on the path or adjacent to a switch-node; that is, we have an embedded caterpillar containing all switch-nodes on the central spine. Existing server-centric DCNs are often such that every server-node is adjacent to at least one switch-node; for such a DCN, if we can find a path that contains every switch-node then we obtain our required caterpillar. Note that whilst finding a caterpillar in an arbitrary graph is NP-hard [42], DCNs are highly structured by design and consequently finding caterpillars should be much easier.

Consider  $HCN(n, h)$ . By identifying a switch-node of  $HCN(n, h)$  and its adjacent server-nodes with a ‘mega-node’, with links between two mega-nodes being inherited in the obvious way, we obtain a WK-recursive network of level  $h - 1$ . As we have already seen, any WK-recursive network of amplitude at least 4 is Hamiltonian-connected [25]; this clearly yields a connected caterpillar in  $HCN(n, h)$  (when  $n \geq 4$ ). Moreover, the Hamiltonian connectivity gives us additional flexibility as to which caterpillar we use to support our virtualization methodology.

However, in order to use caterpillars, derived from such Hamiltonian paths, as embedding vehicles, not only do we need such Hamiltonian paths to exist (as they do in WK-recursive networks), but we need to know how to construct them. As it happens, the proof of Hamiltonian connectivity in [25] is constructive (it uses the recursive-definability of WK-recursive networks); hence, we can build caterpillars in HCN as an aid to embedding virtual clusters.

There is yet more flexibility as regards the availability of myriad paths in HCN along which to embed virtual clusters, for, as we saw earlier, a WK-recursive network

of amplitude at least 5 is pancyclic [24] and so we can use any (maximal length) path on some cycle to embed our virtual clusters; indeed, when the amplitude is at least 6, the WK-recursive network is node-pancyclic [26]. Additionally, and as we mentioned earlier, a WK-recursive network of amplitude  $2n + 1$  has  $n$  link-disjoint Hamiltonian cycles [39]; these link-disjoint cycles might be used to simultaneously embed virtual clusters via a more sophisticated embedding algorithm.

An analogous methodology can be used to embed oversubscribed virtual clusters too: the different virtual clusters within the oversubscribed virtual cluster can be embedded consecutively along a (Hamiltonian) path and the path provides for communication between server-nodes of the same virtual cluster as well as server-nodes of different virtual clusters.

### 7.3 Existing DCNs

The upshot of our discussion is that general ‘symmetry’ properties relating to Hamiltonicity and its variations (such as Hamiltonian connectedness and pancyclicity) are extremely useful properties for various abstractions of a DCN to have in relation to supporting virtualization (via our novel methodology). We now examine this comment in further detail and in relation to other existing DCNs.

Of the existing DCNs mentioned earlier, FiConn, DPillar, HCN, BCN, SWCube, SWKautz, and SWdBruijn are all *dual-port* DCNs, so that every server-node is adjacent to at least one switch-node and at most one server-node (note that dual-port server-centric DCNs can support datacenters built with commodity-off-the-shelf servers which ordinarily only have two NIC ports). Consequently, if we abstract these DCNs so as to form a graph where the nodes are the switch-nodes and where there is an edge joining two nodes if, and only if, there is a path of server-nodes joining the two corresponding switch-nodes in the DCNs, then a Hamiltonian cycle or path in this graph yields a spanning caterpillar in the DCN (note that it might be the case that a pendant server-node in this spanning caterpillar is adjacent to two switch-nodes). Let us refer to the graph abstracted in this way as the *switch-abstraction* of the DCN. As we noted above, the switch-abstraction of  $\text{HCN}(n, h)$ , where  $h \geq 1$ , is a WK-recursive network of amplitude  $n$  and dimension  $h - 1$ .

It is not difficult to see that the switch-abstraction of DPillar contains a wrapped butterfly network; consequently, as a wrapped butterfly network is Hamiltonian [58], we obtain a spanning caterpillar in DPillar. As we remarked earlier, the constructions of SWCube, SWKautz, and SWdBruijn are all of the same type. In order to build SWCube (resp. SWKautz, SWdBruijn), take a generalized hypercube (resp. Kautz graph, de Bruijn digraph) and regard the nodes as switch-nodes with a server-node ‘dividing’ each edge (so as to transform a switch-node-to-switch-node link into a switch-node-to-server-node link followed by a server-node-to-switch-node link; the orientations for the de Bruijn digraph are simply removed from the edges). Consequently, the switch-abstraction of SWCube (resp. SWKautz, SWdBruijn) reverts us back to the generalized hypercube (resp. Kautz graph, de Bruijn digraph). It is well known that a generalized hypercube, a Kautz graph, and a de Bruijn digraph are Hamiltonian (see, for example, [61]) and so we obtain our spanning caterpillar in each of SWCube,



SWKautz, and SWdBruijn. The precise structure of the switch-abstractions of DCell, FiConn, and BCube is not clear and deserves further analysis.

## 8 Conclusions

We have covered a lot of ground in this paper. We have undertaken a thorough consideration of symmetry within server-centric DCNs, motivated by aspects of datacenter usage, namely virtualization and the implementation of communication patterns. We have developed structural metrics, involving recursive-definability, the existence and construction of spanning trees, pancyclicity, and variations in Hamiltonicity, and argued that these metrics imply the suitability of DCNs as regards their capacity for virtualization and to support various communication patterns. Our focus on the underlying server-centric DCN topologies has enabled us to outline a new embedding methodology for server-centric DCNs. Moreover, whilst we have worked with the underlying DCN topology, the structural properties that we have highlighted are such as to make our embedding technique widely applicable and not tied to a specific DCN topology. We have also emphasized the importance and started the development of algebraic techniques to support virtualization and the implementation of communication patterns, and our analysis has resulted in combinatorial abstractions of DCNs as graphs that are directly relevant to virtualization and the implementation of communication patterns.

It is important that a proper understanding of our research is appreciated. What we have done is to highlight aspects of datacenter usage and, from this usage, derived topological properties of DCNs relating to symmetry that will support this usage. We do not claim that these topological properties are definitive as regards DCN design in relation to symmetry or usage, for there are many aspects of datacenter usage that we have not considered. For example, in terms of traffic patterns, we have only considered many-to-many, whereas in reality they are numerous others. What we do claim is that if a DCN possesses the topological properties that we have highlighted here, then it will, in general, be amenable to virtualization and supporting certain communication patterns. Our paper has initiated a closer relationship between DCN design and the usage to which datacenters are put; this relationship has, up until now, not been significantly considered.

Whilst we feel that we have successfully motivated the consideration of aspects of symmetry as regards the design of server-centric DCNs, we also feel that our research has opened up a number of important avenues for further research; indeed, one of the purposes of our paper is to provide the platform for subsequent long-term projects, four of which we highlight below and none of which could be undertaken without recourse to the research in this paper.

### 8.1 Developing server-centric virtualization methodologies

The most obvious direction for research is as regards our outline (in Sect. 7) of a new methodology to embed guest topologies in non-tree-based server-centric DCNs. Note that our proposed new virtualization methodology guarantees locality and has

the potential to improve data collection by limiting the amount of migration traffic generated. We intend to further develop this methodology so that we obtain a fully operational implementation. We also intend to implement virtualization methodologies based on the principles inherent within SecondNet and Oktopus so as to evaluate the different methodologies against each other and across a wide range of server-centric DCNs. This will allow us to empirically evaluate the various graph-theoretic metrics and notions of symmetry that we have proposed. However, this will be a significant undertaking.

Let us highlight here some tasks that need to be accomplished in order to fully develop our proposed server-centric virtualization methodology. Our proposed methodology relies on the existence of (long) paths and cycles in the underlying DCN. Whilst we have exhibited such paths and cycles in the DCN HCN (cf. Fig. 3), we need to find paths and cycles in other server-centric DCNs too. These paths and cycles will need to be algorithmically constructible. Having found our paths and cycles, we need to develop algorithms to ‘stack’ and ‘slide’ virtual clusters within the embedded path (cf. Sect. 7.1), which will be parameterized by the topological structure of the virtual clusters to be embedded and the rate at which migration needs to be undertaken. The resulting algorithmic framework will need to be empirically tested across a range of existing server-centric DCNs. This is entirely feasible, but will be algorithmically involved. In addition, the integration of a virtualization methodology with energy efficiency adds yet more demands.

## 8.2 Devising generic combinatorial constructions to provide symmetry

Our paper has initiated a closer relationship between theoretical computer science (in particular, graph theory) and the design of server-centric DCNs. The design of DCNs has hitherto been undertaken piecemeal, in that different DCNs have been proposed in an ad hoc fashion with no real focus on generic structural properties. We hope that our paper has helped to formalize some of the design methodologies used so far. It is interesting that many of the existing DCNs possess strong relationships with established interconnection networks, e.g. HCN with WK-recursive networks, BCube and SWCube with generalized hypercubes, and also that established combinatorial constructions feature in the construction of these DCNs (albeit implicitly), e.g. the construction of SWCube, SWKautz, and SWdBruijn from the line graphs of generalized hypercubes, Kautz digraphs, and de Bruijn digraphs, respectively. It is also interesting that graph-theoretic abstractions of a DCN, as its clique-abstraction and its switch-abstraction, have key roles to play.

A concerted research effort should now be undertaken to ascertain generic combinatorial constructions that yield new server-centric DCNs and to explore the wider application of these constructions. A recent paper has followed this line of research and proposed the *stellar transformation* which takes any interconnection network and immediately derives a corresponding dual-port server-centric DCN [22] (the construction is similar to those used to build SWCube, SWKautz, and SWdBruijn except that the sub-division of links is with a pair of server-nodes). In [22], an instantiation of this construction using generalized hypercubes is empirically compared with FiConn and DPillar (the results are very promising). However, and pertinent to the research in this

paper, there needs to be a focus on combinatorial constructions that provide support for datacenter usage and communication patterns, together with their integration with energy efficiency; up until now, these aspects have not influenced design at all. In particular, techniques to design new server-centric DCNs encompassing the relevant aspects of symmetry we have highlighted here, such as pancyclicity, Hamiltonicity, and so on, need to be established.

### 8.3 Analysing symmetry within existing DCNs

Existing DCNs are as yet not fully understood in a combinatorial or algebraic sense; we mentioned earlier (in Sect. 6.1.1) our current lack of knowledge as regards, for example, the exact number of server-nodes in FiConn, and also the progress only recently made as regards finding optimal routing algorithms for DPillar and HCN. The key point is that existing DCNs are not well known outside the (engineering-oriented) datacenter community yet they are combinatorial objects that will be of interest to theoreticians and for which theoreticians can prove new properties and algorithms. Such properties and algorithms can then be integrated within more holistic simulations of the DCNs and their practical efficacy evaluated.

As regards the analysis of existing DCNs in relation to the research proposed in this paper, what is required is a theoretical consideration of the degree to which these DCNs adhere to our principles of symmetry; one reason being so that we can see how amenable these DCNs are as regards to supporting the virtualization methodology described above. Of course, this should be combined with extensive simulations of the DCNs under new virtualization methodologies and also under many-to-many traffic patterns (and indeed other traffic patterns arising through datacenter usage). A start has been made in, for example, [57]. As regards simulation, we have developed an open-source, flow-based simulator INRFlow [20] that is specifically designed for flow-based simulation in server-centric DCNs and which can be extended in order to perform these simulations.

### 8.4 Supporting other applications and traffic patterns

We have necessarily had to limit our consideration to virtualization, in terms of applications, and many-to-many traffic patterns, in terms of communication primitives. Even with these limits, we have seen that the situation is complex. However, the reality is that other applications and traffic patterns will impact upon DCN design too, and similar analyses need to be undertaken with respect to alternative usage and traffic. Our focus on virtualization and many-to-many traffic was because of their widespread nature. Of course, we should not expect that alternative usage and traffic frameworks will necessarily yield similar results. As ever in the design of interconnection networks, no matter what the context, there are numerous tensions, with demands often working against each other, and at the heart of the matter is securing a design that can be implemented so that a good overall general performance is secured. In summary,

there is tremendous scope for the fusion of theory and practice in order to better design datacenters and their networks.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

1. Abts D, Felderman B (2012) A guided tour through data-center networking. *ACM Queue* 10(5):1–13
2. Akers SB, Krishnamurthy B (1989) A group-theoretic model for symmetric interconnection networks. *IEEE Trans Comput* 38(4):555–566
3. Al-Fares M, Loukissas A, Vahdat A (2008) A scalable, commodity data center network architecture. In: *Proceedings of ACM SIGCOMM*, pp 63–74
4. Ballani H, Costa P, Karagiannis T, Rowstron A (2011) Towards predictable datacenter networks. In: *Proceedings of ACM SIGCOMM*, pp 242–253
5. Bari MF, Boutaba R, Esteves R, Granville LZ, Podlesny M, Rabbani MG, Zhang Q, Zhani MF (2013) Data center network virtualization: a survey. *IEEE Commun Surv Tutor* 15(2):909–928
6. Barroso LA, Hoelzle U (2009) The datacenter as a computer: an introduction to the design of warehouse-scale machines. Morgan and Claypool, San Rafael
7. Bilal K, Malik SUR, Khalid O, Hameed A, Alvarez E, Wijaysekara V, Irfan R, Shrestha S, Dwivedy D, Ali M, Shahid Khan U, Abbas A, Jalil N, Khan SU (2014) A taxonomy and survey on green data center networks. *Future Gener Comput Syst* 36:189–208
8. Chang Y, Bhuyan LN (1995) A combinatorial analysis of subcube reliability in hypercubes. *IEEE Trans Comput* 44(7):952–956
9. Chen G-H, Hwang S-C, Su M-Y, Duh D-R (1998) A general broadcasting scheme for recursive networks with complete connection. In: *Proceedings of International Conference on Parallel and Distributed Systems (ICPADS)*, pp 248–255
10. Chen C-H, Duh D-R (1994) Topological properties, communication, and computation on WK-recursive networks. *Networks* 24(6):303–317
11. Chen K, Hu C, Xin Z, Zheng K, Chen Y, Vasilakos AV (2011) Survey on routing in data centers: insights and future directions. *IEEE Netw* 25(4):6–10
12. Costa P, Donnelly A, Rowstron A, O’Shea G (2012) Camdoop: exploiting in-network aggregation for big data applications. In: *Proceedings of 9th USENIX Symposium on Networked Systems Design and Implementation*
13. Couto RDS, Secci S, Campista MEM, Costa LHMK (2016) Reliability and survivability analysis of data center network topologies. *J Netw Syst Manag* 24(2):346–392
14. Dai X, Wang JM, Bensaou B (2016) Energy-efficient virtual machines scheduling in multi-tenant data centers. *IEEE Trans Cloud Comput* 4(2):210–221
15. Dally WJ, Towles B (2004) Principles and practices of interconnection networks. Morgan Kaufmann, Los Altos
16. Ding Z, Guo D, Chen X, Luo X (2012) Performing MapReduce on data centers with hierarchical structures. *Int J Comput Commun Control* 7(3):432–449
17. Ding Z, Guo D, Liu X, Luo X, Chen G (2012) A MapReduce-supported network structure for data centers. *Concurr Comput Pract Exp* 24(12):1271–1295
18. Drutskey D, Keller E, Rexford J (2013) Scalable network virtualization in software-defined networks. *IEEE Internet Comput* 17(2):20–27
19. Erickson A, Kiasari A, Navaridas J, Stewart IA (2015) Routing Algorithms for Recursively-defined Data Center Networks. *Proceedings of 13th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pp 84–91
20. Erickson A, Kiasari AE, Pascual Saiz J, Navaridas J, Stewart IA (2016) Interconnection networks research flow evaluation framework (INRFlow) [software]. <https://bitbucket.org/alejandrerickson/inrflow>

21. Erickson A, Kiasari A, Navaridas J, Stewart IA (2017) An optimal single-path routing algorithm in the datacenter network DPillar. *IEEE Trans Parallel Distrib Syst* 28(3):689–703
22. Erickson A, Stewart IA, Kiasari AE, Navaridas J (2017) The Stellar transformation: from interconnection networks to datacenter networks. *Comput Netw* 113:29–45
23. Erickson A, Stewart IA, Pascual JA, Navaridas J (2017) Improved routing algorithms in the dual-port datacenter networks HCN and BCN. *Future Gener Comput Syst* 75:58–71
24. Fernandes R, Friesen DK, Kanevsky A (1994) Embedding rings in recursive networks. In: *Proceedings of 6th IEEE Symposium on Parallel and Distributed Processing*, pp 273–280
25. Fu J (2004) Hamiltonian-connectedness of the WK-recursive network. In: *Proceedings of 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pp 569–574
26. Fang J-F, Huang C-H (2014) On vertex-pancyclicity and edge-pancyclicity of the WK-recursive network. *Inf Sci* 287:131–139
27. Ghemawat S, Gobiuff H, Leung S-T (2003) The Google file system. *ACM SIGOPS Oper Syst Rev* 37(5):29–43
28. Greenberg A, Hamilton JR, Jain N, Kandula S, Kim C, Lahiri P, Maltz DA, Patel P, Sengupta S (2009) VL2: a scalable and flexible data center network. *ACM SIGCOMM Comput Commun Rev* 39(4):51–62
29. Guo C, Lu G, Li D, Wu H, Zhang X, Shi Y, Tian C, Zhang Y, Lu S (2009) BCube: a high performance, server-centric network architecture for modular data centers. *SIGCOMM Comput Commun Rev* 39(4):63–74
30. Guo C, Lu G, Wang HJ, Yang S, Kong C, Sun P, Wu W, Zhang Y (2010) SecondNet: a data center network virtualization architecture with bandwidth guarantees. In: *Proceedings of ACM Conference on Emerging Networking Experiments and Technology*, article no. 15
31. Guo C, Wu H, Tan K, Shi L, Zhang Y, Lu S (2008) DCell: a scalable and fault-tolerant network structure for data centers. In: *Proceedings of IEEE SIGCOMM*, pp 75–86
32. Guo D, Chen T, Li D, Li M, Liu Y, Chen G (2014) Expandible and cost-effective network structures for data centers using dual-port servers. *IEEE Trans Comput* 62(7):1303–1317
33. Guo D, Li D, Wu J, Zhou X (2014) DCube: a family of network structures for containerized data centers using dual-port servers. *Comput Commun* 53:13–25
34. Hammadi A, Mhamdi L (2014) A survey on architectures and energy efficiency in data center networks. *Comput Commun* 40:1–21
35. Heydemann M-C, Ducourthial B (1997) Cayley graphs and interconnection networks. In: Hahn G, Sabidussi G (eds) *Graph symmetry: algebraic methods and applications*, NATO science series C, vol 497. Springer, Dordrecht, pp 167–226
36. Heydemann MC, Meyer JC, Sotteau D (1989) On forwarding indices of networks. *Discrete Appl Math* 23:103–123
37. Hsu L-H, Lin C-K (2008) *Graph theory and interconnection networks*. CRC Press, Boca Raton
38. Huang C-H, Fang J-F (2008) The pancyclicity and the Hamiltonian-connectivity of the generalized base- $b$  hypercube. *Comput Electr Eng* 34(4):63–269
39. Huang C-H, Fang J-F, Yang C-Y (2006) Edge-Disjoint Hamiltonian Cycles of WK-Recursive Networks. In: *Proceedings of 7th International Workshop on Applied Parallel Computing, Lecture Notes in Computer Science*, vol 3732, pp 1099–1104
40. Jerger NE, Peh L-S (2009) On-chip networks. Morgan and Claypool, San Rafael
41. Kachris C, Tomkos I (2012) A survey on optical interconnects for data centers. *IEEE Commun Surv Tutor* 14(4):1021–1036
42. Khosravani M (2011) Searching for optimal caterpillars in general and bounded treewidth graphs. PhD Thesis, University of Auckland
43. Lakshminarayanan S, Jwo J-S, Dhall SK (2003) Symmetry in interconnection networks based on Cayley graphs of permutation groups: a survey. *Parallel Comput* 19(4):361–407
44. Li D, Guo C, Wu H, Tan K, Zhang Y, Lu S, Wu J (2011) Scalable and cost-effective interconnection of data-center servers using dual server ports. *IEEE/ACM Trans Netw* 19(1):102–114
45. Li D, Wu J (2015) On data center network architectures for interconnecting dual-port servers. *IEEE Trans Comput* 64(11):3210–3222
46. Li D, Xu M, Liu Y, Xie X, Cui Y, Wang J, Chen G (2014) Reliable multicast in data center networks. *IEEE Trans Comput* 63(8):2011–2024
47. Li D, Yu J, Yu J, Wu J (2011) Exploring efficient and scalable multicast routing in future data center networks. In: *Proceedings of IEEE INFOCOM*, pp 1368–1376

48. Li D, Zhu J, Wu J, Guan J, Zhang Y (2015) Guaranteeing heterogeneous bandwidth demand in multi-tenant data center networks. *IEEE/ACM Trans Netw* 23(5):1648–1660
49. Liao Y, Yin J, Yin D, Gao L (2012) DPillar: dual-port server interconnection network for large scale data centers. *Comput Netw* 56(8):2132–2147
50. Lin L, Xu L, Zhou S, Wang D (2015) The reliability of subgraphs in the arrangement graph. *IEEE Trans Reliab* 64(2):807–818
51. Liu Y, Muppala JK, Veeraraghavan M, Lin D, Katz J (2013) Data center networks: topologies, architectures and fault-tolerance characteristics. Springer, Berlin
52. Mysore RN, Pamboris A, Farrington N, Huang N, Miri P, Radhakrishnan S, Subramanya V, Vahdat A (2009) PortLand: a scalable fault-tolerant layer 2 data center network fabric. *ACM SIGCOMM Comput Commun Rev* 39(4):39–50
53. Popa L, Ratnaswamy S, Iannaccone G, Krishnamurthy A, Stoica I (2010) A cost comparison of data center network architectures. In: *Proceedings of 6th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, article no. 16
54. Sherwood R, Chan M, Covington A, Gibb G, Flajslik M, Handigol N, Huang T-Y, Kazemian P, Kobayashi M, Naous J, Seetharaman S, Underhill D, Yabe T, Yap K-K, Yiakoumis Y, Zeng H, Appenzeller G, Johari R, McKeown N, Parulkar G (2010) Carving research slices out of your production networks with OpenFlow. *SIGCOMM Comput Commun Rev* 40(1):129–130
55. Vecchia GD, Sanges C (1987) Recursively scalable networks for message passing architectures. In: *Proceedings of International Conference on Parallel Processing and Applications (ICPP)*, pp 33–40
56. Wang B, Qi Z, Ma R, Guan H, Vasilakos AV (2015) A survey on data center networking for cloud computing. *Comput Netw* 91:528–547
57. Wang X, Erickson A, Fan J, Jia X (2015) Hamiltonian properties of DCell networks. *Comput J* 58(11):2944–2955
58. Wong SA (1995) Hamilton cycles and paths in butterfly graphs. *Networks* 26(3):145–150
59. Wu K, Xiao J, Ni LM (2012) Rethinking the architecture design of data center networks. *Front Comput Sci* 6(5):596–603
60. Xu F, Liu F, Jin H, Vasilakos AV (2014) Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions. *Proc IEEE* 102(1):11–31
61. Xu J (2001) Topological structure and analysis of interconnection networks. Kluwer, Dordrecht
62. Xu M, Shang Y, Li D, Wang X (2013) Greening data center networks with throughput-guaranteed power-aware routing. *Comput Netw* 57(15):2880–2899
63. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. *J Internet Serv Appl* 1(1):7–18
64. Zhang Y, Ansari N (2013) On architecture design, congestion notification, TCP incast and power consumption in data centers. *IEEE Commun Surv Tutor* 15(1):39–64
65. Zhao Y, Huang Y, Chen K, Yu M, Wang S, Li D (2015) Joint VM placement and topology optimization for traffic scalability in dynamic datacenter networks. *Comput Netw* 80:109–123
66. Zhu F, Wang H (2016) A modified ACO algorithm for virtual network embedding based on graph decomposition. *Comput Commun* 80:1–15
67. Zhou S (2009) A class of arc-transitive Cayley graphs as models for interconnection networks. *SIAM J Discrete Math* 23(2):694–714