THE Stellar TRANSFORMATION: FROM INTERCONNECTION NETWORKS TO DATACENTER NETWORKS

ALEJANDRO ERICKSON, IAIN A. STEWART, JAVIER NAVARIDAS, AND ABBAS E. KIASARI

Abstract. The first dual-port server-centric datacenter network, FiConn, was introduced in 2009, and there are several others now in existence; however, the pool of topologies to choose from remains small. We propose a new generic construction that dramatically increases the size of this pool by facilitating the transformation of well-studied topologies from interconnection networks, along with their networking properties and routing algorithms, into viable server-centric datacenter network topologies. Under our transformation, numerous interconnection networks yield datacenter network topologies with good, and easily computed, baseline properties. We instantiate our construction so as to apply it to generalized hypercubes and obtain the datacenter networks GQ\textsuperscript{*}. Our construction automatically yields routing algorithms for GQ\textsuperscript{*} and we empirically compare GQ\textsuperscript{*} and its routing algorithms with the established datacenter networks FiConn and D Pillar; this comparison is with respect to network throughput, latency, load balancing, scalability, fault tolerance, and cost. We find that GQ\textsuperscript{*} outperforms both FiConn and D Pillar.

1. Introduction

The digital economy has taken the world by storm and completely changed the way we interact, communicate, collaborate, and search for information. The main driver of this change has been the rapid penetration of cloud computing which has enabled a wide variety of digital services, such as web search and online gaming, by offering elastic, on-demand computing resources to digital service providers. Indeed, the global cloud computing market value is estimated to be in excess of $100 billion [8]. Vital to this ecosystem of digital services is an underlying computing infrastructure based primarily in datacenters [4]. With this sudden move to the cloud, the demand for increasingly large datacenters is growing rapidly [17].

This demand has prompted a move away from traditional datacenter designs based on expensive high-density enterprise-level switches, towards using commodity-off-the-shelf (COTS) hardware. In their production datacenters, major operators have primarily adopted (and invented) ideas similar to Fat-Tree [2], Portland [31] and VL2 [15]; on the other hand, the research community (several major operators included) maintains a diverse economy of datacenter architectures and designs in order to meet future demand [13,17,19,29,34,35]. Indeed, the “switch-centric” datacenters currently used in production datacenters have inherent scalability limitations, and are by no means a low-cost solution [7,18,20,28].

One approach intended to help overcome these limitations is “the server-centric” architecture; typical examples are DCell [17] and BCube [16]. By offloading the task of routing packets to servers, this architecture leverages the typically low utilisation of CPUs in datacenters to manage network communication. This can reduce both the number of switches used in a network and the capabilities required of them. In particular, the switches route only locally, to their neighbouring servers, and therefore have no need for large or fast routing tables. Thus a server-centric datacenter network (DCN) is potentially both cheaper to operate and to build (see [32] for a related discussion). Furthermore, using servers, which are highly programmable, to route packets, rather than switches, which have proprietary software and limited programmability, will potentially accelerate research innovation [26].

Several server-centric topologies further restrict themselves to requiring at most two ports per server [18,24,27]. The dual-port restriction is motivated by the fact that many COTS servers presently available for purchase, as well as servers in existing datacenters, have two NIC ports (a primary and a backup port). Dual-port server-centric DCNs are able to utilise such servers without modification, thus making it possible to use some of the more basic equipment (available for purchase or from existing datacenters) in a server-centric DCN, thereby reducing the cost of building one. Our research is within the context of dual-port server-centric DCNs.
In this paper, a server-centric DCN is modelled by a graph with two types of nodes, server-nodes and switch-nodes, and edges that represent physical wired links in the DCN, with the property that the switch-nodes form an independent set. This restriction, that no two switch-nodes are connected by an edge, arises from the situation that the switches in a server-centric DCN act as non-blocking “dumb” crossbars. Dual-port server-centric DCN topologies have the restriction that server-nodes are of degree at most 2.

The server-centric DCN architecture provides a versatile design space, as regards the network topology, evidenced perhaps by the sheer number of fairly natural constructions proposed from 2008 to the present. On the other hand, this pool is small relative to the number of interconnection networks found in the literature; i.e., highly structured graphs with good networking properties. One of the challenges of identifying an interconnection network suitable for conversion to a DCN topology, however, lies in the fact that the literature on interconnection networks is focused primarily on graphs whose nodes are homogeneous. Some server-centric DCN topologies arise largely from outside this literature, e.g., DCell [17] and FiConn [24], whilst others arise from transformations of well-known interconnection networks, e.g., BCube [16] and DPillar [27].

The transformations used in BCube and DPillar take advantage of certain substructures in the base graph in order to create a server-centric DCN that inherits some of the base graph’s networking properties, such as low diameter and fault-tolerant routing algorithms. The limitation, of course, is that not every interconnection network has the required sub-structures (cliques and bicliques, in the cases of BCube and DPillar). New methods of transforming interconnection networks into server-centric DCNs may therefore greatly enlarge the DCN design space by lowering the structural requirements of potential base graphs.

In this paper we propose stellar networks for transforming interconnection networks as base graphs into dual-port server-centric DCNs. Under the transformation, the edges of the base graph are replaced with 3-paths, so that the nodes of the base graph become the switch-nodes, and the added nodes, interior to the 3-paths, become the server-nodes of the stellar network (see Fig. 3). By requiring very little of the base graph, in terms of structure, the stellar construction greatly increases the pool of interconnection networks that can potentially serve to design DCN topologies.

We validate our construction in three ways: first, we prove that various good networking properties of the base graph are preserved under the stellar network transformation; second, we build a library of interconnection networks that suit the transformation; third, we empirically evaluate GQ*, an instantiation of stellar networks whose base graphs are generalized hypercubes.

We also validate the effectiveness of transforming previous work on routing algorithms for the base graphs [10] into a routing algorithm, GQSRouting, for GQ*. GQSRouting provides excellent performance in terms of average path-lengths and connectivity in the presence of failures. In particular, we find that when 10% of the links in the network have failed, GQSRouting provides around 95% connectivity and it generates paths that are, on average, only around 10% longer than the shortest paths found by a breadth-first search. Shortest-path routing is achievable in a fault-free network, of course; however, GQSRouting paths are potentially 2% longer than the shortest paths due to technicalities of the implementation.

Our empirical evaluation of GQ* includes a comparison with well-established prior work on dual-port server-centric DCNs. We use a comprehensive set of performance metrics including network throughput, latency, load balancing capability, scalability, and fault tolerance to compare GQ* to the well-established DCNs FiConn and DPillar. We find that GQ* comprehensively outperforms FiConn (except as regards amortised component-cost per server, where they are approximately equal) and that it offers overall better performance than that of DPillar, as well as requiring roughly half the number of switch-nodes for a DCN with a similar number of server-nodes.

The rest of the paper is organized as follows. In the next section, we give an overview of the design space for dual-port server-centric DCNs, and related work, before defining our new generic construction in Section 3 and presenting its theoretical advantages. In Section 4 we define our instantiation so as to generate the DCNs GQ*. Section 5 and Section 6 describe our experimental set-up and results. We close the paper with some concluding remarks and suggestions for future work in Section 7.

2. The Dual-Port Server-Centric
We take a primarily mathematical view of datacenters in order to systematically identify potential DCN topologies by using graphs to model only the major hardware components of the DCN; servers, switches, and the cables that interconnect them.

A dual-port server-centric DCN can be built from COTS servers, each with (at most) two network interface cards NIC ports, dumb “crossbar” switches, and the cables that connect these hardware components together. We define the capability of a dumb crossbar-switch (henceforth referred to as a switch) as being able to forward an incoming packet to a single port requested in the packet header, and handle all such traffic in a non-blocking manner. In reality, such a switch will receive packets destined only for servers directly attached to it and handle the requests by retrieving their addresses from a very small forwarding table. There is no purpose, therefore, in having two switches in the network connected by a cable, so we assume this situation does not occur. We model such a DCN with a graph $G$ with two types of node: server-nodes of degree at most two, representing servers; and an independent set of switch-nodes, representing switches.

2.1. Designing DCNs with good networking properties. There are well-established performance metrics for DCNs and their routing algorithms such as network throughput, latency, load balancing capability, scalability, fault tolerance, and cost to build. Networks that perform well with respect to these or related metrics are said to have good networking properties. Maintaining a diverse pool of potential topologies with good networking properties gives DCN designers greater flexibility. There is indeed already such a pool, comprising over 50 years of research on interconnection networks and related combinatorics, and it is precisely from here that the switch-centric DCN fabrics of layer-2 switches in fat-trees and related topologies [2, 23] have been adapted.

Adapting interconnection networks to build server-centric DCNs, which may have two different types of nodes, however, is more complicated. For example, BCube [16] is built from a generalized hypercube (see Definition 1) by replacing the edges of certain cliques, each with a switch-node connected to the nodes of the clique. In doing so, BCube inherits well-known routing algorithms for generalized hypercubes, as well as mean-distance, fault tolerance, and other good networking properties. DPillar, which we discuss in detail in Section 2.2, is built in a similar manner, using bicliques in wrapped butterfly networks. The presence of these cliques and bicliques are inherent in the definitions of generalized hypercubes and wrapped butterfly networks, respectively, but are not properties of interconnection networks in general. Furthermore, the dual-port property of DPillar is not by design of the construction, but is a result of the fact that each node in a wrapped butterfly is in exactly two maximal bicliques.

In order to capitalise on a wide range of interconnection networks, for the purpose of server-centric DCN design, we must devise construction methods, like those of BCube and DPillar, that do not impose such severe structural requirements on the interconnection network used as a base graph.

2.2. Related work. We briefly survey the origins of the dual-port server-centric DCNs proposed thus far [18][24][27], referring the reader to the original publications for definitions of topologies not given below. The topologies HCN and BCN [18] are built by combining a 2-level DCell with another network, later discovered to be related to WK-recursive networks [11][36]. FiConn [24] is an adaptation of DCell and is unrelated to any particular interconnection network. BCCC is a tailored construction related to BCube based on cube-connected-cycles and generalized hypercubes [26]. DPillar’s origins were discussed above [27]. Finally, SWKautz and SWCube [25] employ a subdivision rule similar to ours, but the focus of their work is not on the benefits of subdividing interconnection networks as much as it is on the evaluation of those two particular network topologies.

In Section 6 we compare $GQ^*$ to FiConn and DPillar. The rationale for using these DCNs in our evaluation is that they are good representatives of the spectrum of server-centric dual-port DCNs, which were mentioned in Section 2.1. FiConn is a good example of DCNs that include both server-to-server and server-to-switch connections and are somewhat unstructured, whereas DPillar is server-node symmetric and features only server-to-switch connections. In addition, FiConn is arguably unrelated to any previously known interconnection network topology (barring DCell), whilst DPillar is built from, and inherits some of the properties

---

2 A set of nodes that are pairwise interconnected.

3 Meaning that for every pair $(u, v)$ of server-nodes there is an automorphism of the network topology that maps $u$ to $v$. 

3
of, the wrapped butterfly. Various other dual-port server-centric DCNs lie somewhere between these two extremes. Notice that neither FiConn nor DPillar can be described as an instance of our generalised construction: FiConn has some server-nodes whose only connection is to a switch-node, and in DPillar each server-node is connected to 2 switch-nodes. We now provide the constructions of the families of DCNs FiConn and DPillar.

2.3. The construction of FiConn. We start with FiConn, the first dual-port DCN to be proposed, and so typically considered the baseline DCN. For any even \( n \geq 2 \), \( \text{FiConn}_{k,n} \) \[24\] is a recursively-defined DCN where \( k \) denotes the level of the recursive construction and \( n \) the number of server-nodes that are directly connected to a switch-node (so, all switch-nodes have degree \( n \)). \( \text{FiConn}_{0,n} \) consists of \( n \) server-nodes and one switch-node, to which all the server-nodes are connected. Suppose that \( \text{FiConn}_{k,n} \) has \( b \) server-nodes of degree 1 (\( b = n \) when \( k = 0 \)). In order to build \( \text{FiConn}_{k+1,n} \), we take \( \frac{n}{2} + 1 \) copies of \( \text{FiConn}_{k,n} \) and for every copy we connect one server-node of degree 1 to each of the other \( \frac{n}{2} \) copies (these additional links are called level \( k \) links). The actual construction of which server-node is connected to which is detailed precisely in \[24\] (\( \text{FiConn}_{2,4} \), as constructed in \[24\], can be visualised in Fig. 1); in particular, there is a well-defined naming scheme where server-nodes of \( \text{FiConn}_{k,n} \) are named as specific \( k \)-tuples of integers. In fact, although it is not made clear in \[24\], there is a multitude of connection schemes realising different versions of FiConn.

2.4. The construction of DPillar. The DCN \( \text{DPillar}_{n,k} \) \[27\], where \( n \) denotes the number of ports of a switch-node and where \( k \) denotes the level of the recursive construction of the DCN, can be imagined as \( k \) columns of server-nodes and \( k \) columns of switch-nodes, arranged alternately on the surface of a cylindrical pillar (see as an example \( \text{DPillar}_{6,3} \) in Fig. 2). Each server-node in some server-column is adjacent to 2 switch-columns, in different adjacent switch-columns. Each server-column has \( \left( \frac{n}{2} \right)^k \) server-nodes, named as \( \{0, 1, \ldots, \frac{n}{2} - 1\}^k \), whereas each switch-column has \( \left( \frac{n}{2} \right)^{k-1} \) switch-nodes, named as \( \{0, 1, \ldots, \frac{n}{2} - 1\}^{k-1} \).

Fix \( c \in \{0, 1, \ldots, k - 1\} \). The server-nodes in server-columns \( c, c+1 \in \{0, 1, \ldots, k - 1\} \) (with addition modulo \( k \)) are arranged into \( \left( \frac{n}{2} \right)^{k-1} \) groups of \( n \) server-nodes so that in server-columns \( c \) and \( c+1 \),
the server-nodes in group \((u_{k-1}, \ldots, u_{c+1}, u_{c-1}, \ldots, u_0)\) are the server-nodes named \\{\((u_{k-1}, \ldots, u_{c+1}, i, u_{c-1}, \ldots, u_0) : i \in \{0, 1, \ldots, \frac{n}{2} - 1\}\\}. The adjacencies between switch-nodes and server-nodes are such that any server-node in group \((u_{k-1}, \ldots, u_{c+1}, u_{c-1}, \ldots, u_0)\) in server-columns \(c\) and \(c+1\) is adjacent to the switch-node of name \((u_{k-1}, \ldots, u_{c+1}, u_{c-1}, \ldots, u_0)\) in switch-column \(c\).

While the above networks have merit, it is clear that a generic method of transforming interconnection networks into dual-port server-centric DCNs has not previously been proposed and analysed. Having justified the value in studying the dual-port restriction, and having discussed the benefits of tapping into a large pool of potentially useful topologies, we proceed by presenting our construction in the next section.

### 3. Stellar networks: a new generic construction

In this section we present our generic method of transforming interconnection networks into potential dual-port server-centric DCN topologies. We describe how networking properties of the DCN as well as routing algorithms are inherited from the base graph, and identify a preliminary pool of interconnection networks that suit the stellar transformation.

Let \(G = (V, E)\) be any non-trivial connected graph, which we call the base graph of our construction. The stellar DCN \(G^*\) is obtained from \(G\) by placing 2 server-nodes on each link of \(G\) and identifying the original nodes of \(G\) as switch-nodes (see Fig. 3). We use the term ‘stellar’ as we essentially replace every node of \(G\) and its incident links with a ‘star’ subnetwork consisting of a hub switch-node and adjacent server-nodes. Clearly, \(G^*\) has \(2|E|\) server-nodes and \(|V|\) switch-nodes, with the degree of every server-node being 2 and the degree of every switch-node being identical to the degree of the corresponding node in \(G\).

We propose placing 2 server-nodes on every link of \(G\) so as to ensure: uniformity in that every server-node is adjacent to exactly 1 server-node and exactly 1 switch-node; that there are no links incident only with switch-nodes (as this would violate a restriction of server-centric DCN topologies, discussed in Section 2); and that we can incorporate as many server-nodes as needed within the construction (subject to other conditions).

Any DCN in which every server-node is adjacent to exactly 1 server-node and 1 switch-node and where every switch-node is only adjacent to server-nodes can be realised as a stellar DCN \(G^*\), for some graph
$G$. In addition, the transformation can be applied to any (non-trivial connected) base graph; that is, the transformation does not rely on any non-trivial structural properties of the base graph.

3.1. Properties of stellar DCNs. The principal decision that must be taken when constructing a stellar DCN is in choosing an appropriate base graph $G = (V, E)$. The good networking properties discussed in Section 2.1 are underpinned by several graph theoretical properties that are preserved under the stellar transformation: low diameter, high connectivity, and efficient routing algorithms in the base graph $G$ translate more-or-less directly into good networking properties of the stellar graph $G^*$. The DCN designer, having specific performance targets in mind, can use this information to facilitate the selection of a base graph $G$ that meets the requirements of the desired stellar DCN.

A key observation is as regards the transformation of paths in $G$ to paths in $G^*$. As is usual in the analysis of server-centric DCNs (see, e.g., [16–18, 24]), we measure a server-node-to-server-node path $P$ by its hop-length, defined as one less than the number of server-nodes in $P$. Accordingly, we prefix other path-length-related measures with hop-, as in the hop-distance between two vertices and the hop-diameter of a server-centric DCN topology. Let $G = (V, E)$ be a connected graph, and let $u, v \in V$. Let $u^*, v^*$ be the vertices of $G^*$ corresponding to $u$ and $v$, respectively. Let $u', v'$ be server-node neighbours of $u^*$, $v^*$, respectively. Each $(u, v)$-path $P$ in $G$, of length $m$, corresponds uniquely to a $(u^*, v^*)$-path in $G^*$ with $2m - 1$, $2m$, or $2m + 1$ server hops. The details are straightforward.

The transformation of paths in $G$ to paths in $G^*$ is the basis for the transfer of potentially useful substructures in $G$, and thereby good networking properties, so as to yield good DCN properties. Any useful substructure in $G$, such as a spanning tree, a set of disjoint paths, or a Hamiltonian cycle, corresponds uniquely to a substructure in $G^*$. Swathes of research papers have uncovered these substructures in interconnection networks, and the stellar construction facilitates their usage in dual-port server-centric DCNs. It would be impossible to cover this entire topic, but we describe how a few of the more commonly sought-after substructures behave under the transformation.

Foremost are node-disjoint paths, associated with fault-tolerance. As the degree of any server-node in $G^*$ is 2, one cannot hope to obtain more than 2 node-disjoint paths joining any 2 distinct server-nodes of $G^*$. However, a set of $c$ internally disjoint $(u, v)$-paths in $G$ corresponds uniquely to a set of $c$ internally disjoint $(u^*, v^*)$-paths in $G^*$, where $u, v, u^*, v^*, u', v'$ are defined as above. This provides a set of $c$ $(u', v')$-paths in $G^*$, called parallel paths, that are internally disjoint apart from possibly $u^*$ and $v^*$ (see Fig. 3). It is trivial to show that the minimum number of parallel paths between any pair of server-nodes, not connected to the same switch-node, in $G^*$ is equal to the vertex connectivity of $G$.

![Figure 3. Transforming 4 internally disjoint paths from $u$ to $v$ in $G$, (left), into 4 parallel paths from server-node $u'$ to server-node $v'$ in $G^*$, (right).](image)
By this logic it is easy to see that a set of $c$ edge-disjoint $(u, v)$-paths in $G$ becomes a set of $c$ internally server-node-disjoint $(u', v')$-paths in $G^*$, with $u, v, u^*, v^*, u', v'$ defined as above; we shall call these server-parallel paths. The implication is that any two of these paths share only the ports incident to the links $(u', u^*)$ and $(v^*, v')$, so a high value of $c$ may be leveraged to alleviate network traffic congestion as well as fortify the network against server failures.

We summarise the relationship between properties of $G$ and $G^*$ that we have discussed so far in Table 1.

<table>
<thead>
<tr>
<th>property</th>
<th>$G = (V, E)$</th>
<th>$G^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>$</td>
<td>V</td>
</tr>
<tr>
<td>(switch-)node degree</td>
<td>$d$</td>
<td>$d$</td>
</tr>
<tr>
<td>links</td>
<td>$</td>
<td>E</td>
</tr>
<tr>
<td>mean distance</td>
<td>$x$</td>
<td>$2x - 1 \leq x \leq 2x + 1$</td>
</tr>
<tr>
<td>diameter</td>
<td>$D$</td>
<td>$2D - 1, 2D$, or $2D + 1$</td>
</tr>
<tr>
<td>connectivity</td>
<td>$\kappa$</td>
<td>$\kappa$ parallel paths</td>
</tr>
<tr>
<td>edge connectivity</td>
<td>$\gamma$ edge-disjoint paths</td>
<td>$\gamma$ server-parallel paths</td>
</tr>
</tbody>
</table>

On a more abstract level, consider any connected spanning substructure $H$ of $G$, such as a Hamiltonian cycle or a spanning tree. Let $H^*$ be the corresponding substructure in $G^*$ and observe that each edge of $G$ not contained in $H$ corresponds to two adjacent server-nodes in $G^*$ not contained in $H^*$. On the other hand, every server-node not in $H^*$ is exactly one hop away from a server node that is in $H^*$; so within an additive factor of one hop, $H^*$ is just as “useful” in $G^*$ as $H$ is in $G$. By the same principle, non-spanning substructures of $G$, such as those used in one-to-many, many-to-many, and many-to-one communication patterns are also useful in $G^*$.

The power of the stellar transformation becomes even more apparent when combining the above two properties, namely the proximity of server-nodes to substructures $H^*$ and the transformation of edge-disjointness in $G$ into server-node disjointness in $G^*$. We shall illustrate this with spanning trees but the following holds in a more general setting. Let $S$ and $T$ be a pair of totally independent spanning trees; that is, $S$ and $T$ are edge-disjoint, and for any pair of vertices $u, v$, the $(u, v)$-path in $S$ is internally vertex-disjoint from the $(u, v)$-path in $T$. Totally independent spanning trees have applications in efficient fault-tolerant broadcasting and have been studied in several of the interconnection networks we mention below in Section 8.2 (e.g., see [21]). Let $S^*, T^*$ be the structures corresponding to $S, T$, respectively, within $G^*$, and let $u^*, v^*, u', v'$ be defined as above. Our earlier discussion on spanning substructures yields that $S^*$ and $T^*$ are server-node-disjoint, and for any pair of server-nodes $(u', v')$, there is a pair of parallel $(u', v')$-paths, one through $S^*$ and the other through $T^*$, and in addition, $S^*$ and $T^*$ are server-node- (and edge-) disjoint.

We stop short of fully enumerating structural properties and their uses in DCNs while stressing that this is an important consideration in choosing a base graph for a stellar construction.

3.1.1. Transformation of routing algorithms in $G^*$. A routing algorithm on an interconnection network $G$ is effectively concerned with the efficient computation of communication substructures. For example, in the case of unicast routing, we may compute one or more $(u, v)$-paths (and route packets over them), or for a broadcast we may compute one or more spanning trees, or simply flood the network. Routing algorithms can be executed at the source node or in a distributed fashion, and they can be deterministic or non-deterministic; whatsoever the process, the resulting output is a communication structure over which the servers can send packets. In the previous subsection we discussed the correspondence between communication substructures in $G$ and those in $G^*$; here, we observe that, furthermore, any routing algorithm on $G$ can be simulated on $G^*$ with the same time complexity. We leave the details to the reader, but see Section 4.2 where we illustrate this further with an instantiation of stellar networks.

3.2. Suitable base graphs for the stellar construction. We validate our claim that many interconnection networks suit the stellar construction by establishing several that are representative of the properties one would look for. The first goal is typically to identify networks with a suitable combination of node-degree and
network size. Today’s DCN COTS switches have up to tens of ports, with 48 being typical, while conceivable (but not necessarily in production) sizes of DCNs range from tens of servers up to, perhaps, 5 million in the near future. While there is certainly no lack of research on low and fixed-degree families of interconnection networks, such as hypercubes (the starred-hypercube $Q^*_n$, has over 1 million server-nodes and degree 16) and cube-connected cycles (which are regular, of degree 3), we exhibit some families that also contain members of higher degree, relative to the number of nodes. It would be impossible to analyse the many structural features and networking properties of these interconnection networks here; however, there is no need to do so since we have shown in general how $G^*$ inherits structural features and networking properties from the base graph $G$.

Circulant graphs have been studied extensively in a networking context, where they are often called multi-loop networks; for example, see the survey by Hwang [22]. Let $S$ be a set of integers, called jumps, with $1 \leq s \leq \left\lfloor \frac{n}{2} \right\rfloor$ for each $s \in S$, where $N \geq 2$. A multi-loop network $L(N; S)$ has node set $\{0, 1, \ldots, N - 1\}$, where node $i$ is connected to nodes $i \pm s \pmod{N}$, for each $s \in S$ (a directed variant exists as well, but in the context of DCNs all links are considered to be bidirectional). The degree of a multi-loop network is approximately $2 |S|$, providing great flexibility, in general. Results on circulant graphs show that certain low- and fixed-radix circulants have good networking properties [5, 37], as do certain denser ones. Thus, we propose that circulant graphs are worth exploring further as base graphs for stellar DCN topologies.

The wrapped butterfly, from which DPillar$_{n,k}$ is built, is usually viewed as an indirect network, but work by Fu and Chau and others [14, 38] provides results on shortest-path routing algorithms, connectivity, graph embeddings, and edge-disjoint spanning trees. For even $n$ and $k \geq 3$, the stellar wrapped-butterfly $B^*(n, k)$ has $k(n/2)^k$ switch-nodes and $nk(n/2)^k$ server-nodes. The diameter of $B(n, k)$ is $\left\lceil \frac{3k}{2} \right\rceil$, and small values of $k$ yield $B^*(n, k)$ with hundreds of thousands of server-nodes. For example, $n = 16$ gives 24576, 262144, 2621440 server-nodes with $k = 3, 4, 5$, respectively. Note that DPillar$_{n,k}$ is not a subdivision of $B(n, k)$, and therefore does not necessarily benefit from the same properties of the stellar construction as $B^*(n, k)$ would, such as edge-disjoint spanning trees [38]. Singla et al. proposed the use of pseudo-random regular graphs (RRGs) as the basis for a layer-2 switch fabric in the DCNs Jellyfish [35]. While we note the important practical point that large DCNs made from RRGs are likely impossible to package efficiently (with short wires, for example), Jellyfish makes an important theoretical contribution. Likewise, a stellar RRG would benefit from the same expansion properties present in RRGs, and would certainly be a good candidate for further exploration if the packaging problem could be addressed.

The de Bruijn digraph $DB(n, k)$ is a graph on $(n/2)^k$ nodes, each labelled with $s_{k-1}s_{k-2}\cdots s_0$, where $0 \leq s_i \leq n/2$ and for each $0 \leq i < k$. There is a directed edge from $s_{k-1}s_{k-2}\cdots s_0$ to $s_{k-2}s_{k-3}\cdots s_{0}0$, for each $0 \leq \alpha < n/2$. De Bruijn digraphs have also been studied as undirected networks [33] and have a number of good networking properties. The underlying undirected simple graph of $DB(n, k)$ is not regular but nearly so, where most of its nodes are degree $n$, and with some of degree $n + 1$ or $n - 2$, when $n^k$ is sufficiently large. Thus $DB^*(n, k)$ has around $n^{k+1}$ server-nodes and has switch-node degree at most $n$. For example, with $(n, k) = (16, 3)$ we have around 65536 server-nodes.

The arrangement graph $A(n, k)$ has $n!/(n-k)!$ nodes and is of regular degree $k(n-k)$. Arrangement graphs [1] include the well-known star graphs [1] and there is considerable flexibility in their degree and network size. One of several configurations of similar size, $A^*(15, 3)$, has 98280 server-nodes, switch-nodes of degree 36, and diameter at most 9.

We conclude this subsection by defining generalized hypercubes [6], which we will look at in detail in Section 3.3.

**Definition 1.** The generalized hypercube $GQ_{k,n}$ of dimension $k$ and radix $n$ has node-set $\{0, 1, \ldots, n - 1\}^k$ and there is a link joining two nodes if, and only if, the names of the two nodes differ in exactly one coordinate.

### 3.3. Implementing stellar DCNs

Just as stellar DCN topologies inherit relevant mathematical properties of the base graph, so can they reuse most of the software stack in other server-centric architectures, such as BCube [3] or DCell [24], as there is no fundamental difference in their nature. However, some topology-related parts such as server names and routing algorithms need to be adapted. We discuss this in detail for a specific instantiation of stellar graphs in Section 4.
Implementing the software suite from scratch would require a software infrastructure that supports through-server end-to-end communications. This could be implemented either on top of the transport layer (TCP) so as to simplify development, since most network-level mechanisms (congestion control, fault tolerance, quality of service) would be provided by the lower layers. Alternatively, it could be implemented on top of the data-link layer to improve the performance, since a lower protocol stack will result in faster processing of packets. The latter would require a much higher implementation effort in order to deal with congestion and reliability issues. At any rate, the design and development of a software suite for server-centric DCNs is outside the scope of this paper, but may be considered in the future.

4. The stellar DCNs $GQ^*$

For the remainder of the paper we instantiate our stellar construction with particular base graphs, namely generalized hypercubes $GQ_{k,n}$, defined in Definition 1 to obtain the family of DCNs $GQ^*_{n,k}$. In this section we develop the basic and structural properties (see Table 2) as well as routing algorithms for $GQ^*$.

4.1. Naming the nodes of $GQ_{n,k}$. By construction, the switch-nodes of $GQ_{n,k}$ adopt the names of the corresponding nodes in $GQ_{k,n}$; thus the set of switch-nodes of $GQ_{n,k}$ is $\{0, 1, \ldots, n-1\}^k$. Let $\mathbf{u}, \mathbf{v} \in \{0, 1, \ldots, n-1\}^k$ be a pair of adjacent nodes in $GQ_{k,n}$. We name the server-nodes $u$ and $v$ on the 3-path $\mathbf{u}, u, v, \mathbf{v}$ in $GQ^*_{n,k}$ by $(\mathbf{u}, \mathbf{v})$ and $(\mathbf{v}, \mathbf{u})$, respectively.

4.2. Routing. The stellar construction allows us to transform existing routing algorithms for the base graph $GQ_{k,n}$ into routing algorithms for $GQ^*_{n,k}$. We describe this process using the routing algorithms for $GQ_{k,n}$ surveyed by Young and Yalamanchili 10.

Let $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ and $\mathbf{x} = (x_0, x_1, \ldots, x_{k-1})$ be two distinct nodes of $GQ_{k,n}$. The basic routing algorithm for $GQ_{k,n}$ is dimension-order (or e-cube) routing where the path from $\mathbf{u}$ to $\mathbf{v}$ is constructed by sequentially replacing each $u_i$ by $v_i$, for some predetermined ordering of the coordinates $0, 1, \ldots, k-1$. As a result of the discussion in Section 3.1.1 dimension-order routing translates into a shortest-path routing algorithm for $GQ^*_{n,k}$ with the same time complexity, namely, $O(k)$.

We introduce a fault-tolerant mechanism called intra-dimensional routing by allowing the path to replace $u_i$ by $v_i$ in two steps, using a local proxy, rather than in one step, as described in dimension-order routing. Suppose, for example, that one of the edges in the dimension order route from $\mathbf{u}$ to $\mathbf{v}$ is faulty, say, the one from $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ to $\mathbf{x} = (v_0, u_1, \ldots, u_{k-1})$, where $u_0$ and $v_0$ are distinct. In this case we can try to hop from $\mathbf{u}$ to $(x_0, u_1, \ldots, u_{k-1})$, where $u_0 \neq x_0 \neq v_0$, and then to $\mathbf{x}$.

Inter-dimensional routing is a routing algorithm that extends intra-dimensional routing so that if intra-dimensional routing fails, because a local proxy within a specific dimension cannot be used to re-route round a faulty link, an alternative dimension is chosen. For example, suppose that in $GQ_{k,n}$ intra-dimensional routing has successfully built a route over dimensions 1 and 2 but has failed to re-route via a local proxy in dimension 3. We might try and build the route instead over dimension 4 and then return and try again with dimension 3. Note that if a non-trivial path extension was made in dimension 4 then this yields an entirely different locality within $GQ_{k,n}$ when trying again over dimension 3.

4.3. GQSRouting. We implement the most extensive fault-tolerant, inter-dimensional routing algorithm possible, called $GQSRouting$ for $GQ^*_{n,k}$, whereby we perform a depth-first search of the dimensions, and we use intra-dimensional routing to cross each dimension wherever necessary (and possible). In addition, if $GQSRouting$ fails to route directly in this fashion then it attempts four more times to route (as above) from the source to a randomly chosen server, and from there to the destination. We have chosen to make this extensive search of possible routes in order to test the maximum capability of $GQSRouting$; however, we expect that in practice the best performance will be obtained by limiting the search in order to avoid certain worst case scenarios. The details of $GQSRouting$ add little to the present discussion, however, they can be found in the software release of INRFlow 12 (see Section 5.0).

Lastly, while our algorithm computes the route recursively, it is easy to see that it can be distributed across the network with a small amount of extra header information attached to a path-probing packet such as the one described in 24 for implementing Traffic Aware Routing (TAR) in FiConn.
Table 2. Basic properties of GQ*\(_{k,n}\) for \(n > 2\) (GQ*\(_{k,2}\) has diameter \(2k\)).

<table>
<thead>
<tr>
<th>servers</th>
<th>switches</th>
<th>ports</th>
<th>diameter</th>
<th>parallel paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>((k(n - 1)n^k))</td>
<td>(n^k)</td>
<td>((k(n - 1)))</td>
<td>(2k + 1)</td>
<td>((k(n - 1)))</td>
</tr>
</tbody>
</table>

5. Methodology

The networking properties discussed in Section 2.1 guide our evaluation methodology. They are network throughput, latency, load balancing capability, scalability, fault tolerance, and cost to build. These real-world performance metrics are reflected in properties of our graph models of DCNs, and in this section we explain how we use ABT, distance metrics, and node disjoint paths, combined with a selection of traffic patterns in order to evaluate the performance of DCNs and routing algorithms.

How a multidimensional set of metrics is interpreted depends on what purpose a network is used for and which metrics are prioritised. First of all, we take the position, similar to Popa et al. [32], that the cost of a network is of universal importance. No matter what purpose a network is intended for, the primary objective is to maximise the return on the cost of a DCN. While there are several elements that factor into the cost of a DCN, including operational costs, our concern is with the cost of purchasing and installing the components we are modelling: servers, switches, and cables. We perform what is effectively the inverse of the evaluation from Popa et al. [32]. That is, we normalise with respect to cost and proceed by both quantitatively and qualitatively interpreting the resulting multidimensional comparison. This more flexible evaluation allows our results to be interpreted in a broader context, where there can be diverse priorities on network performance. We then focus on 4 carefully chosen DCNs, namely GQ*\(_{3,10}\), GQ*\(_{4,6}\), FiConn\(_{2,24}\), and DPillar\(_{18,4}\) and study these DCNs in more detail. We have selected these DCNs as their properties are relevant to the construction of large-scale DCNs: they each have around 25,000 servers and use switches of around 24 ports. Table 3 details some of their topological properties.

Table 3. Basic properties of the selected DCNs.

<table>
<thead>
<tr>
<th>topology</th>
<th>GQ*(_{3,10})</th>
<th>GQ*(_{4,6})</th>
<th>FiConn(_{2,24})</th>
<th>DPillar(_{18,4})</th>
</tr>
</thead>
<tbody>
<tr>
<td>servers</td>
<td>27,000</td>
<td>25,920</td>
<td>24,648</td>
<td>26,244</td>
</tr>
<tr>
<td>switches</td>
<td>1,000</td>
<td>1,296</td>
<td>1,027</td>
<td>2,916</td>
</tr>
<tr>
<td>switch radix</td>
<td>27</td>
<td>20</td>
<td>24</td>
<td>18</td>
</tr>
<tr>
<td>links</td>
<td>81,000</td>
<td>77,760</td>
<td>67,782</td>
<td>104,976</td>
</tr>
<tr>
<td>diameter</td>
<td>7</td>
<td>9</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>parallel paths</td>
<td>27</td>
<td>20</td>
<td>unknown</td>
<td>9 (from [27])</td>
</tr>
</tbody>
</table>

5.1. Network cost. The amortised component-cost per server-node of FiConn is marginally less than that of GQ*, which is less, in turn, than that of DPillar. For certain plausible relative costs of switches, servers, and cables, the relative cost of DCNs GQ*, FiConn, and DPillar can be roughly approximated by the number of server-nodes they contain. We derive expressions for the amortised component-cost per server-node.

First, we make the assumption that the cost of a switch-node is proportional to its radix; this is justified in Popa et al. [32] for switches of radix of up to around 100 – 150 ports. Let \(c_s\) be the cost of a server, let \(c_p\) be the cost of a switch-port, and let \(c_c\) be the average cost of a cable. We make the simplifying assumption that the average cost of a cable \(c_c\) is uniform across DCNs with \(N\) server-nodes within the families GQ*, FiConn, and DPillar, and furthermore, that the average cost of a cable connected only to servers is similar to that of a cable connected to a switch. Thus, the cost of a DCN GQ* with \(N\) server-nodes is \(N(c_p + c_c + c_s + c_c/2)\); the cost of a DCN FiConn\(_{k,n}\) with \(N\) server-nodes is \(N(c_p + c_c + c_s + c_c/2 - c_c/2^{k+1})\), since it contains \(N/2^k\) server-nodes of degree 1 [24]; the cost of a DCN DPillar with \(N\) server-nodes is \(N(2c_p + c_c + c_s)\). Second, we express \(c_p = \rho c_s\) and \(c_c = \gamma c_s\) in terms of the cost per server, \(c_s\), so that the amortised costs per server-node, respectively, become \(N c_s (\rho + \gamma + 1 + \gamma/2)\); \(N c_s (\rho + \gamma + 1 + \gamma/2 - \gamma/2^{k+1})\); and, \(N c_s (2(\rho + \gamma) + 1)\). A rough estimate for realistic values of \(\rho\) is in the range \([0.01, 0.1]\), and realistic values for \(\gamma\) are in the range \([0.01, 6]\). We choose the ranges conservatively because there is great variation in the cost of components, for example, between copper and optical cables, as well as how we account for the labour involved in installing...
them. Thus we normalise with respect to the amortised component cost per server-node in GQ\(^\star\), letting \(c_s(\rho + \gamma + 1 + \gamma/2) = 1\) and plot amortised component costs per server-node against \(\gamma\) in Fig. 4 for each \(\rho \in \{0.01, 0.02, 0.4, 0.8, 1.6\}\). The upshot is that for the specific choices of \(\rho\) and \(\gamma\) mentioned above, DPillar could be up to 20\% more expensive and FiConn around 4\% less expensive than a DCN GQ\(^\star\) with around the same number of servers. Perhaps the most realistic values of \(\rho\) and \(\gamma\), however, yield a DPillar that is only about 10\% more expensive.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure4.png}
\caption{The amortised component costs per server of FiConn, and DPillar, relative to that of GQ\(^\star\), for \(\rho \in \{0.01, 0.02, 0.4, 0.8, 1.6\}\). Higher \(\rho\) yields a higher cost for DPillar.}
\end{figure}

5.2. Hop-distance metrics. The number of servers a packet flow needs to travel through significantly affects the flow's latency. In addition, for each server on the path, the compute and memory overheads are impacted upon: in a server-centric DCN (with currently available COTS hardware), the whole of the protocol stack, up to the application level, needs to be processed at each server which can make message transmission noticeably slower than in a switch-centric network where lower layers of the protocol stack are employed and use optimised implementations.

The paths over which flows travel are computed by routing algorithms, and it may not be the case that shortest-paths are achievable by known routing algorithms and without global fault-knowledge; large-scale networks like DCNs are typically restricted to routing algorithms that use only local knowledge of fault locations. As such, the performance of the routing algorithm is perhaps more important than the hop-diameter or mean hop-distance of the topology itself. Therefore we use distance-related metrics that reveal the performance of topology and routing algorithm combined, routed hop-diameter, routed mean hop-distance, as well as for the topology alone (where appropriate), hop-diameter, and mean hop-distance (see Section 2.1). This allows us to assess both the potential of the topologies and the actual performance that can be extracted from them when implemented with current technology.

5.3. Aggregate bottleneck throughput. The aggregate bottleneck throughput (ABT) is a metric introduced in [16] and is of primary interest to DCN designers due to its suitability for evaluating the worst-case throughput in the all-to-all traffic pattern, which is significant in the context of DCNs (See Section 5.5). The reasoning behind ABT is that the performance of an all-to-all operation is limited by its slowest flow, i.e., the flow with the lowest throughput. The ABT is defined as the total number of flows times the throughput of the bottleneck flow – the link sustaining the most flows. Formally, the ABT of a network of size \(N\) is equal to \(N(N-1)b/F\), where \(F\) is the number of flows in the bottleneck link and \(b\) is the bandwidth of a link.

In our experiments the bottleneck flow is determined experimentally using actual routing functions (see Section 5.6); this is atypical of ABT calculations (e.g., see [28]) as it requires the implementation of the actual routing functions, but it facilitates a more realistic evaluation. In our paper, the ABT is measured using GQSRouting for GQ\(^\star\), TOR for FiConn and SP for DPillar, assuming \(N(N-1)\) flows and a bandwidth of 1 unit per directional link, where \(N\) is the number of servers. Since datacentres are most commonly used as a stream processing platform and are therefore bandwidth limited, this is an extremely important performance
metric in the context of DCNs. Since ABT is only defined in the context of all-to-all communications, for other traffic patterns we will focus on the number of flows in the bottleneck as an indicator of congestion propensity.

5.4. Fault tolerance. High reliability is of utmost importance in datacenters, as it impacts upon the business volume that can be attracted and sustained. When scaling out to tens of thousands of servers or more, failures are common, with the mean-time between failures (MTBF) being as short as hours or even minutes. As an example, consider a datacenter with 25,000 servers, 1,000 switches and 75,000 links, each with an optimistic average lifespan of 5 years. Based upon a very rough estimate that the number of elements divided by the average lifespan results in the numbers of failures per day, the system will have an average of about 13 server faults per day, 40 link faults per day and 1 switch fault every 2 days. In other words, failures are ubiquitous and so the DCN should be able to deal with them in order to remain competitively operational. Any network whose performance degrades rapidly with the number of failures is unacceptable, even if it does provide the best performance in a fault-free environment.

We investigate how network-level failures affect \textit{routed connectivity}, defined as the proportion of server-pairs that remain connected by a path computable by a given routing algorithm, as well as routed mean hop-distance. Our study focuses on uniform random link failures, in particular, because both server-node and switch-node failures induce link failures, and also because the sheer number of links (and NICs) in a DCN implies that link-failures will be the most common event. A more detailed study of failure events will be conducted in follow-up research, in which we will consider correlated link, server-node, and switch-node failures. We consider failure configurations with up to a 15% network degradation, where we randomly select 15% of the links to have a fault, with uniform probability. Furthermore, we consider only bidirectional failures, i.e., where links will either work in both directions or in neither of them. The rationale for this is that the bidirectional link-failure model is more realistic than the unidirectional one: failures affecting the whole of a link (e.g., NIC failure, unplugged or cut link, switch port failure) are more frequent than the fine-grained failures that would affect a single direction. In addition, once unidirectional faults have been detected they will typically be dealt with by disabling the other direction of the failed link (according to the IEEE 802.3ah EFM-OAM standard).

5.5. Traffic patterns. We now describe the traffic patterns used in our evaluation, the primary one being the all-to-all traffic pattern. All-to-all communications are relevant as they are intrinsic to MapReduce, the preferred paradigm for data-oriented application development; see, for example, [10, 28, 39]. In addition, all-to-all can be considered a worst-case traffic pattern for two reasons: \(a\) the lack of spatial locality, and \(b\) the high levels of contention for the use of resources.

Our second set of experiments focuses on specific networks hosting around 25,000 servers and evaluates them with a wider collection of traffic patterns. Apart from all-to-all, we also consider the three other traffic patterns: many all-to-all, butterfly, and random. In many all-to-all the network is split into disjoint groups of a fixed number of servers with servers within a group performing an all-to-all operation. Our evaluation shows results for groups of 1,000 servers but these are consistent with those for groups of sizes 500 and 5,000. This workload is less demanding than the system-wide all-to-all, but can still generate a great deal of congestion. It aims to emulate a typical tenanted cloud datacenter, in which there are many applications running concurrently. We assume a typical topology-agnostic scheduler and randomly assign servers to groups. The butterfly traffic pattern is a logarithmic implementation of collective operations (such as all-to-all or allreduce) in which each server only communicates with other servers at distance \(2^k\), for each \(k \in \{0, \ldots, \lfloor \log(N) \rfloor - 1\}\) (see [20] for more details). This workload significantly reduces the overall utilization of the network when compared with the all-to-all-based traffic patterns and aims to evaluate the behavior of the networks when the traffic pattern is well-structured. Finally, we consider a random traffic pattern in which we generate one million flows (we also studied other numbers of flows, but the results are very similar to those with one million flows). For each flow, source and destination are selected at random following a uniform distribution. These additional collections of experiments provide further insights into the performance achievable with each of the networks and allow a more detailed evaluation.

5.6. Software tools. Our software tool, Interconnection Networks Research Flow Evaluation Framework (INRFlow) [12] is designed for testing large-scale systems such as DCNs with tens or hundreds of thousands
of nodes, which would prohibit the use of of packet-level simulations. The results obtained from INRFlow inform a detailed evaluation within the intended scope of our paper, which proposes the use of the stellar construction in general.

INRFlow is capable of evaluating network topologies in two ways. The first is by undertaking a BFS for each server node; this allows us to compute the hop-length of the shortest path between any two server-nodes and also to examine whether two server-nodes become disconnected in the presence of link failures. As we have noted in Section 5.2, results on shortest paths are of limited use when not studied in conjunction with a routing algorithm. Thus, INRFlow also provides path and connectivity information about a given routing algorithm. By this we obtain experimental results using “Traffic Oblivious Routing” \( TOR \) for FiConn (see [24]), “Single Path” \( SP \) and “Multi Path” \( MP \) (for the study of fault-tolerance) routing for DPillar (see [27]) and our newly proposed \( GQSRouting \) (described earlier) so as to obtain a more realistic measure of performance. The operation of the tool is as follows: for each flow in the workload, it computes the route using the required routing algorithm and updates link utilization accordingly. Then it reports a large number of statistics of interest, including the metrics discussed above.

We use the algorithm \( TOR \) for FiConn instead of the routing algorithm “Traffic Aware Routing” (\( TAR \)) for two reasons. First, \( TOR \) yielded better performance for all-to-all traffic patterns than \( TAR \) in the evaluations in [24]. Second, \( TAR \) is a distributed heuristic algorithm devised so as to improve network load balancing with bursty and irregular traffic patterns and was neither optimised nor tested on outright faulty links. In addition, \( TAR \) computes paths that are 15–30% longer in these scenarios than \( TOR \) does. A comparison between \( GQSRouting \) and \( TAR \) would be inherently biased against FiConn, so we use \( TOR \) in the relevant parts of our evaluation, and, since \( TOR \) is not fault-tolerant, we compare faulty DCNs \( GQ^* \) to FiConn directly, using BFS.

Error bars. The absence of error bars in our evaluation is by design. In our paper, random sampling occurs in two different ways: the first is where a random set of faulty links is chosen and properties of the faulty topology are plotted, as in \( ???????? \); the second is with regards to randomised traffic patterns, as in \( ???????? \). For each set of randomised link failures we plot statistics, either on connectivity or path length, for the all-to-all traffic pattern (i.e., the whole population of server-node-pairs).

In \( ???????? \) we sample the mean of two statistics over the set of all possible sets of \( m \) randomised link failures based on only one trial for each network and statistic, and therefore it does not make sense to compute estimated standard error for these plots. The true error clearly remains very small, however, because of the high level of uniformity of the DCNs we are studying, including the non-symmetric DCN FiConn. The uniformity effectively simulates a large number of trials, since, for each choice of faulty link there are hundreds or thousands of other links in the DCN whose failure would have almost exactly the same effect on the overall experiment. Quantifying this error is outside the scope of our paper; however, it is evident from the low amount of noise in our plots that the true error is negligible in the context of the conclusions we are making. ??? sample flows to find the mean number of links with a certain proportion of utilisation, and to find the mean hop-lengths of the flows. Our sample sizes, given in Section 5.5 are exceedingly large for this purpose, and thus error bars would be all but invisible in these plots. We leave the explicit calculation to the reader.

6. Evaluation

In this section we perform an empirical evaluation of the DCNs \( GQ^* \), comparing the performance with that of the DCNs FiConn and DPillar using the methodology and definitions detailed in Section 5.

6.1. Scalability and cost. We evaluate how well \( GQ^* \) scales in comparison with FiConn and DPillar.

We first consider ABT vs the number of servers in each network. Fig. 5 shows that ABT scales much better in \( GQ^* \) than in FiConn. For the largest systems considered, \( GQ^* \) supports up to around three times the ABT of FiConn\( _{3,n} \). The difference between \( GQ^* \) and FiConn\( _{2,n} \) is not as large but is still substantial. We can see that although \( GQ^* \) networks are constructed using far fewer switches and links than DPillar, their maximum sustainable ABT is similar and, indeed, \( GQ^* \) networks with \( k = 2 \) and \( k = 3 \) consistently outperform the DCNs DPillar. Because of it’s relation to network throughput, the ABT is often a primary consideration in DCN design, and these results show that \( GQ^* \) is highly competitive with respect to this metric.
Figure 5. The ABT using GQSRouting, TOR, and DPillarSP.

Figure 6. The ABT in terms of network cost for GQSRouting, TOR, and DPillarSP, where a DCN DPillar is 110% the cost of a DCN GQ with the same number of server nodes, whilst FiConn is 98% of the cost of GQ. Network cost is normalised by the amortised component cost per server in GQ.

Figure 7. Routed mean hop-distances for GQ*/GQSRouting, FiConn/TOR, and DPillar/DPillarSP.

Fig. 6 shows a plot of ABT vs network cost, under the most plausible assumptions discussed in Section 5.1, that the amortised cost of components for DPillar is around 10% more, and that of FiConn is around 2%
less than those of GQ*. Under these conditions, we can see that the plotted GQ*’s outperform nearly every instance of FiConn and DPillar.

The increase in ABT for GQ*_{n,k} as k decreases follows from two facts. First, for a fixed number of servers, reducing k results in an increased switch radix, which translates into higher locality. Second, reducing k results in lower routed mean hop-distance (see Fig. 7), which lowers the total utilization of the DCN and, when combined with good load balancing properties, yields a bottleneck link with fewer flows.

This latter phenomenon also appears in Fig. 7 where the lower k is, the lower the routed mean hop-distance. At any rate, we can see that the routed mean hop-distances obtained in the different topologies increase very slowly with network size and are, of course, bounded by the routed hop-diameter, which is dependent on k for all 3 topologies: 2k+1 for GQ*/GQSRouting; 2k−1 for DPillar/DPillarSP; and 2k+1−1 for FiConn/TOR. The ‘exponential nature’ of FiConn discourages building this topology for any k larger than 2.

Although we forgo a simulation of packet injections, our experiment does allow for a coarse-grained latency analysis. Network latency is brought on by routing packets over long paths and spending additional time processing (e.g., buffering) the packets at intermediate nodes, due to network congestion. These scenarios have various causes, but they are affected generally by the DCNs ability to simultaneously balance network traffic and route it over short paths efficiently. Figs. 6–7 show that GQSRouting scales well with respect to load balancing (high ABT), and routed mean hop-distance, from which we infer that in most situations GQ*_{3,n} has lower latency than GQ*_{4,n} and all FiConn DCNs, and likely performs similar to DPillar_{n,3}.

In summary, GQ* has better scalability properties than FiConn and, in certain important respects, also outperforms the denser DCNs DPillar. This is especially true for the ABT which, as discussed in Section 5.3, is a performance metric of primary interest in the context of datacenters.

6.2. Fault tolerance. A priori, GQ* features a provably high number of parallel paths and server-parallel paths compared to FiConn and DPillar of similar size (see Table 3). Thus, if GQSRouting utilises these paths, we expect strong performance in degraded networks. Fig. 8 shows the routed connectivity under failures of GQ*/GQSRouting and DPillar/DPillarMP. The plot indicates that DPillarMP underutilises the network, since the unrouted connectivity of DPillar (not plotted) is slightly stronger than that of GQ*. This highlights the fact that choosing a topology with good algorithms and other non-trivial networking properties should be a top consideration in DCN design, and it motivates a more detailed evaluation of GQSRouting (and indeed fault-tolerant routing for DPillar).

Note that the evaluation of DPillar/DPillarMP in Liao et al. 27 is with respect to server faults, in which the performance of DPillarMP looks stronger than it does in our experiments with link-failures. This is because the failed servers do not send messages and therefore do not factor into the connectivity of the faulty DPillar.

With FiConn not having a fault-tolerant algorithm comparable to GQSRouting (see Section 5.6), we plot the unrouted connectivity of GQ* with that of FiConn, using BFS, in Fig. 9. To our knowledge there is no fault-tolerant routing algorithm for FiConn that achieves anything close to the performance of BFS; however, Fig. 11 shows that GQSRouting very nearly achieves the (optimum) unrouted connectivity of GQ*.

We have shown that GQ* and GQSRouting are very competitive in terms of fault-tolerance.

6.3. Assessment of GQSRouting. We assess the performance of GQSRouting by comparing it with optimum performance, computed by a BFS, which finds a shortest-path if it exists. Notice that since dimensional routing is a shortest path algorithm on GQ_{k,n}, it is straightforward to modify GQSRouting so as to be a shortest path algorithm on GQ*_{k,n}; however, due to simplifications in our implementation there is a discrepancy of about 2% between shortest paths and GQSRouting in a fault-free GQ*.

Of interest to us here is the relative performance of GQSRouting and BFS in faulty networks. Fig. 10 plots the routed and unrouted mean hop-distances in networks with a 10% link failure rate, where the difference between GQSRouting and BFS in mean hop-distance is close to 10%. This is a reasonable overhead for a fault-tolerant routing algorithm, especially given its high success rate at connecting pairs of servers in faulty networks. Fig. 11 plots the unrouted connectivity, which is optimum and achieved by a BFS, and the routed

5Routed and unrouted data computed for other DCNs GQ* was very similar and is not plotted for the sake of presenting clearer plots.
connectivity, achieved by GQRouting, for the same, 10%, failure rate.\footnote{GQRouting appears to be better than BFS for certain numbers of servers, but this is because the faults were generated randomly for each test.} As implemented, GQRouting is optimised for maintaining connectivity at the cost of routing over long paths if necessary. A different mix of features might reduce the 10% gap in Fig. 10 and increase the gap in Fig. 11. In any case, the performance of GQRouting is very close to the optimum.

\footnote{GQRouting appears to be better than BFS for certain numbers of servers, but this is because the faults were generated randomly for each test.}
6.4. Detailed evaluation of large-scale DCNs. We turn our attention to 4 concrete instances of the topologies and their routing algorithms: GQ\(^\star\),\textsubscript{3,10}, GQ\(^\star\),\textsubscript{4,6} with GQSRouting, FiConn\textsubscript{2,24} with TOR, and DPillar\textsubscript{18,4} with DPillarSP. They were chosen for the reasons outlined in Section 5, and we give their basic properties in Table 3. In this subsection we refer to each topology and its routing algorithm as a unit.

![Figure 11](image)

**Figure 11.** Routed (GQSRouting) and unrouted connectivity of GQ\(^\star\) with 10% link failures.

![Figure 12](image)

**Figure 12.** Relative number of flows in the bottleneck for the different traffic patterns, normalised to FiConn/TOR.

![Figure 13](image)

**Figure 13.** Routed hop-distance for the different traffic patterns.

Fig. 12 shows the number of flows in the bottleneck for the different traffic patterns considered in our study. We can see that these results confirm the previous results in that GQ\(^\star\) can outperform well-known DCNs as
the two instances studied here can reduce significantly the number of flows in the bottleneck, thus improving the overall throughout. The only exception is DPillar\(_{18,4}\) with the butterfly traffic pattern. The rationale for that is that the butterfly pattern matches perfectly the DPillar topology and, thus, it allows a very good balancing of the network, reducing the flows in the bottleneck. For the rest of the patterns, DPillar\(_{18,4}\) is clearly the worst performing in terms of this metric. Fig. 13 shows the routed mean hop-distance for the different patterns and topologies and shows that DPillar, due to the higher number of switches, can generally reach its destination using the shortest paths. Note that even with the clear advantage of having higher availability of shorter paths, DPillar\(_{18,4}\) still features the worst number of flows in the bottleneck and, therefore, is the most prone to congestion. On the other hand GQ\(^*\)\(_{4,6}\), which has the longest paths, has the second lowest number of flows in the bottleneck, after GQ\(^*\)\(_{3,10}\).

**Figure 14.** Normalised histogram of number of flows for three DCNs under the all-to-all traffic pattern. The mean number of flows per link are 89567, 101953, 84615, and 141187, for GQ\(^*\)\(_{3,10}\), GQ\(^*\)\(_{4,6}\), FiConn\(_{2,24}\) and DPillar\(_{18,4}\), respectively. Connecting lines drawn for clarity.

**Figure 15.** Distribution of number of flows per link for random traffic pattern. Not plotted are 52488 unused links in DPillar and 6162 unused links in FiConn.

The reason for these surprising results can be found if we look more closely at the number of flows supported by each link in the topology. For example, the histogram in Fig. 14 shows that both of the plotted GQ\(^*\)s are much better balanced than FiConn\(_{2,24}\) and DPillar\(_{18,4}\) under the all-to-all traffic pattern: in GQ\(^*\)\(_{3,10}\) all of the links sustain between 60,000 and 100,000 flows; similarly, in GQ\(^*\)\(_{4,6}\) the links have between 80,000 and 120,000 flows. Nearly 25% of the links in FiConn\(_{2,24}\), however, have less than 40,000 flows, whereas the other 75% of the links have between 80,000 and 140,000 flows. This is even worse for DPillar\(_{18,4}\) in which half of the links have more than 100,000 flows while the other half are barely used.
The imbalances present in FiConn$^{2,24}$ and DPillar$^{18,4}$ result in parts of the networks being underutilised and other parts being overly congested.

A more detailed distribution obtained using the random traffic pattern is shown in Fig. 15. There, we can see how both GQ’s are clearly better balanced than FiConn$^{2,24}$, as the latter has two pinnacles: one of low-load with about 30% of the links and another of high-load with the rest of the links. We can also see that choosing the bottleneck link as the figure of merit is reasonable as it would yield similar results as if we had chosen the peaks in the plot.

As in Section 6.1, we can infer that GQ$^{*,3,10}$ will provide better latency figures than GQ$^{*,4,6}$ and FiConn$^{2,24}$ as it has fewer flows in the bottleneck link and shorter paths. The shorter paths in DPillar$^{18,4}$ do suggest that with low-intensity communication workloads it should have lower latency than GQ$^{*,3,10}$, but since DPillar$^{18,4}$ is poorer at balancing load than GQ$^{*,3,10}$, we can infer that it may have higher latency under high-intensity communication workloads such as the ones typically used in datacentres.

7. Conclusion

This paper proposes the stellar construction for transforming interconnection networks and their network- ing properties into dual-port server-centric DCNs. This construction provides several nice properties which simplify network design. The most important of them is that it can be applied to any base graph and the knowledge we have about this base graph can be reused (e.g., routing, theoretical distance properties, node-disjoint paths and so on) to build improved DCNs. We explicitly identified several families of interconnection networks whose degree and link-density are suited to the stellar construction.

As a case study we instantiated our construction with generalized hypercubes and their well-studied fault-tolerant algorithms as the base graphs to obtain the stellar DCNs GQ$^*$ and the fault-tolerant routing algorithm GQSRouting, which we compared to the DCNs and routing algorithms FiConn/TOR and DPillar/DPillarSP/DPillarMP. Our experiments show that GQ$^*/GQSRouting$ consistently outperforms FiConn/TOR in every performance metric considered in this study (except amortised component cost per server, where they are approximately equal) and that it offers overall better performance than that of DPillar/DPillarSP/DPillarMP, as well as requiring roughly half the number of switches to be built. In particular GQ$^*$ can provide higher throughput and better load balancing than FiConn and DPillar for several traffic patterns of interest, including all-to-all traffic. GQSRouting routes traffic over slightly longer paths than DPillarSP implying that with low traffic congestion, DPillar will exhibit lower latency, but with high traffic congestion GQ$^*$ will outperform DPillar in terms of latency as well, due to DPillar’s susceptibility to traffic congestion. GQSRouting also far exceeds the performance of DPillarMP in degraded DCNs, and under reasonable assumptions, the amortised component cost per server of DPillar is around 10% higher than that of GQ$^*$.

Comparing GQSRouting against the intrinsic properties of the topology, GQ$^*$, we find our implementation of GQSRouting finds paths that are within 2% of the optimal length (0% is realistically possible) and around 10% for degraded networks with 10% faulty links. It is a relatively small overhead for our routing algorithm which achieves very high connectivity—typically 95% connectivity for 10% uniform random link failures.

There are a number of open questions arising from this paper that we will investigate in the future. A non-comprehensive list is as follows: analyse the practicalities (floor planning, wiring, availability of local routing) of packaging the DCNs GQ$^*$; perform a broader evaluation using a higher number of DCN architectures and traffic models; refine GQSRouting to produce minimal paths for fault-free networks and compare its performance with the near-optimal algorithm used in this paper; apply the stellar transformation to other well-understood topologies; and finally, explore the effect of the stellar construction on formal notions of symmetry in the base graph.

Acknowledgements

This work has been funded by the Engineering and Physical Sciences Research Council (EPSRC) through grants EP/K015680/1 and EP/K015699/1. The authors gratefully acknowledge this support, as well as the valuable recommendations made by the anonymous referees.
References


20


School of Engineering and Computing Sciences, Durham University, Science Labs, South Road, Durham DH1 3LE, U.K.

School of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, U.K.