Accepted Manuscript

Improved routing algorithms in the dual-port datacenter networks HCN and BCN

Alejandro Erickson, Iain A. Stewart, Jose A. Pascual, Javier Navaridas

PII: S0167-739X(17)30013-4
DOI: http://dx.doi.org/10.1016/j.future.2017.05.004
Reference: FUTURE 3452

To appear in: *Future Generation Computer Systems*

Received date: 9 January 2017
Revised date: 4 April 2017
Accepted date: 5 May 2017

Please cite this article as: A. Erickson, I.A. Stewart, J.A. Pascual, J. Navaridas, Improved routing algorithms in the dual-port datacenter networks HCN and BCN, *Future Generation Computer Systems* (2017), http://dx.doi.org/10.1016/j.future.2017.05.004

Improved Routing Algorithms in the Dual-port Datacenter Networks HCN and BCN

Alejandro Erickson[a], Iain A. Stewart[a,*], Jose A. Pascual[b], Javier Navaridas[b]

[a]*School of Engineering and Computing Sciences, Durham University, Science Labs, South Road, Durham DH1 3LE, U.K.*
[b]*School of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, U.K.*

Abstract

We present significantly improved one-to-one routing algorithms in the datacenter networks HCN and BCN in that our routing algorithms result in much shorter paths when compared with existing routing algorithms. We also present a much tighter analysis of HCN and BCN by observing that there is a very close relationship between the datacenter networks HCN and the interconnection networks known as WK-recursive networks. We use existing results concerning WK-recursive networks to prove the optimality of our new routing algorithm for HCN and also to significantly aid the implementation of our routing algorithms in both HCN and BCN. Furthermore, we empirically evaluate our new routing algorithms for BCN, against existing ones, across a range of metrics relating to path-length, throughput, and latency for the traffic patterns all-to-one, bisection, butterfly, hot-region, many-all-to-all, and uniform-random, and we also study the completion times of workloads relating to MapReduce, stencil and sweep, and unstructured applications. Not only do our results significantly improve routing in our datacenter networks for all of the different scenarios considered but they also emphasise that existing theoretical research can impact upon modern computational platforms.

Keywords: datacenters; datacenter networks; HCN; BCN; one-to-one routing; WK-recursive networks; performance metrics.

1. Introduction

Datacenters are becoming pervasive within the global computational infrastructure and the sizes of these datacenters are expanding rapidly, with some of the largest operators managing over a million servers across multiple datacenters. As to how these servers are interconnected via the *datacenter network* (*DCN*) is a fundamental issue, the consideration of which involves a mix of mathematics, computer science, and engineering. Moreover, just as with the design of interconnection networks for distributed-memory multiprocessors or networks-on-chips, there is no 'silver bullet' solution, for there is a wide range of design parameters to consider, some of which are conflicting.

The traditional architecture of a DCN is 'switch-centric' whereby the primary structure is a topology (usually tree-based) of switches with the switches possessing interconnection intelligence. The DCNs Fat-Tree [2], VL2 [12], and Portland [21] are typical of such DCNs. A more recent and alternative architecture is 'server-centric' whereby the interconnection intelligence resides within the servers and the switches are dumb crossbars (so, there are no switch-to-switch links). The DCNs DCell [14], FiConn [18], BCube [13], MDCube [27], HCN and BCN [15], and GQ* [9] are typical of server-centric DCNs.

The server-centric architecture possesses a number of advantages when compared with the more traditional switch-centric architecture: tree-based switch-centric DCNs tend to be such that 'root' switches quickly become a bottleneck; the underlying topologies of server-centric topologies are better suited to support traffic patterns prevalent in datacenters (such as one-to-all and all-to-all); the switches in server-centric DCNs can be chosen to

*Corresponding author
Email addresses: alejandro.erickson@gmail.com (Alejandro Erickson), i.a.stewart@durham.ac.uk (Iain A. Stewart), jose.pascual@manchester.ac.uk (Jose A. Pascual), javier.navaridas@manchester.ac.uk (Javier Navaridas)

be commodity switches as they require no intelligence; and multiple network interface controller (NIC) ports on servers in server-centric DCNs can be utilized so that more varied topologies can be constructed (see, for example, [5, 15, 19] for more information).

Whilst multiple NIC ports can be used when building server-centric DCNs, commodity servers usually only have a small number of NIC ports, often only two. This can be problematic as a primary aim of DCN design is to incorporate a large number of servers within the datacenter. For example, when one builds the DCNs DCell, BCube, and MDCube, one finds that the number of NIC ports required increases as the number of servers rises. On the other hand, FiConn and GQ*, for example, is such that no matter how many servers there are, each server needs only two NIC ports; such server-centric DCNs are referred to as *dual-port*.

Motivated by the need to limit the number of NIC ports on servers (so that commodity servers might be used), Guo *et al.* introduced and evaluated the dual-port DCNs HCN and BCN [15]. The general construction is that the DCN HCN is a recursively-defined family of networks, with the DCN BCN built using (copies of) the DCN HCN by including an additional layer of interconnecting links. After defining the DCNs HCN and BCN, Guo *et al.* developed a number of routing algorithms (including one-to-one, multipath, and fault-tolerant algorithms) and evaluated HCN and BCN, primarily in comparison with FiConn and according to a number of basic metrics.

We pursue the analysis of the DCNs HCN and BCN in this paper. In particular, we present significantly improved one-to-one routing algorithms in both HCN and BCN, in that our routing algorithms result in much shorter paths than those in [15] (our analysis is both theoretical and empirical). We also present a much tighter analysis of HCN and BCN by observing that there is close relationship between the DCN HCN and the interconnection networks known as *WK-recursive networks*, which originated in [25] and which have been well studied as general interconnection networks. We use existing theoretical results concerning WK-recursive networks to develop our routing algorithms and prove the optimality of our new routing algorithm for HCN (in terms of path length), as well as to significantly aid the implementation of our routing algorithms in both HCN and BCN. Not only do we develop routing algorithms for HCN and BCN that are theoretical improvements over existing routing algorithms but we undertake an extensive empirical evaluation of our algorithms, against existing ones, for DCNs of a range of realistic sizes, under a range of traffic patterns and workloads, and across a range of metrics. In particular, we consider metrics relating to hop-length, throughput, and latency for the ('static') traffic patterns all-to-one, bisection, butterfly, hot-region, many-all-to-all, and uniform-random, and we also study the completion times of ('dynamic') workloads relating to MapReduce, stencil and sweep, and unstructured applications, where these workloads have data associated with flows and might involve some causality between flows. We also study how the connection rule used to build BCN out of copies of HCN, of which there are currently two in the literature (though potentially many more), impacts upon the resulting DCN BCN, in terms of the above empirical analysis. Our simulations are undertaken with our own purpose-built flow-based simulator `INRFlow` [8]. A novel aspect of our simulations is that whereas the 'static' simulation of routing algorithms on the above traffic patterns is the norm within the server-centric research community, `INRFlow` allows us to simulate our routing algorithms on the above 'dynamic' workloads (insofar as we are aware, this paper contains the first such 'dynamic' simulations on server-centric DCNs).

Our results are extremely encouraging, for we almost universally obtain improvements. Not only do we obtain theoretically-improved algorithms but our empirical analysis suggests that there are significant gains to be made by the practical deployment of our new routing algorithms in HCN and BCN. For example, when compared with the routing algorithm *BdimRouting* for BCN (from [15]), our primary new routing algorithm for BCN, namely $NewBdimRouting_{\gamma}$, achieves hop-length savings for all DCNs studied and across all traffic patterns, averaging at around a 25% improvement. What is more, a practical version of $NewBdimRouting_{\gamma}$, namely $NewBdimRouting_1$, where we curtail the inherent search for shorter paths within $NewBdimRouting_{\gamma}$, is shown to give a performance comparable with that of $NewBdimRouting_{\gamma}$. Our algorithm $NewBdimRouting_1$ also achieves a significant improvement in both throughput and latency when compared with *BdimRouting* in the different scenarios: as regards throughput, on average this improvement is by 36% and 55% for the two throughput metrics we consider; and as regards latency,

on average this improvement is by 10%. Our algorithm $NewBdimRouting_1$ also obtains improvements for all the different 'dynamic' workloads mentioned above.

This paper is structured as follows. In the next section, after detailing the essential concepts of server-centric DCNs, we give precise definitions of the DCN HCN, and exhibit the link with WK-recursive networks, and the DCN BCN. In Section 3, we develop new one-to-one routing algorithms for HCN, prove their optimality, and explain how they can be very easily implemented. Our new one-to-one routing algorithms for BCN are developed in Section 4. In Section 5, we explain the framework for and reasoning behind our experiments, and in Section 6, we supply and evaluate the results we obtain. Our conclusions and directions for further research are presented in Section 7. A preliminary version of this paper where the analysis only considered HCN appeared as [24].

2. Server-centric datacenter networks

In this section we define the graph-theoretic abstractions that we use to obtain our results on server-centric DCNs. A server-centric DCN is built from commodity off-the-shelf (COTS) switches and servers, interconnected by cable links. It is distinguished from other types of datacenters in that very low capability is required of the switches, which act as simple, non-blocking crossbars, and any routing algorithms and network protocols are implemented within the servers. Thus, we abstract a server-centric DCN as a graph $G = (S \cup W, E)$, where $u \in S$ is a *server-node*, representing a server, and $w \in W$ is a *switch-node*, representing a switch, and each link in E represents a physical link of the DCN. The only requirement, imposed by the simplicity of the switches we are modelling, is that no two switch-nodes are connected by a link; as such, $E \cap (W \times W) = \varnothing$. As we shall see, our DCNs come in parameterized families. Henceforth, we use the term DCN to refer to both a family member and the family itself.

A routing algorithm[1] takes a pair of server-nodes, (src, dst), as input and outputs a path, P, in G from src to dst. The *path-length* of P is equal to the number of links P contains, and the *hop-length* of P is equal to the number of hops it contains, where a *hop* is a link joining two server-nodes or a path of path-length 2 from a server-node to another server-node through a switch-node. Hop-length is the primary distance-related performance metric used in evaluations of server-centric DCNs (see, *e.g.*, HCN and BCN [15], DCell [14], FiConn [18], BCube [13], MDCube [27], and GQ* [9]), for the reason that packets must travel up and down the protocol stack of each intermediate server to reach the service that will route them to the next server, rendering negligible the time spent at each switch. We work with hop-length in this paper.

2.1. The DCN HCN

Let us define the DCN HCN(n, h) from [15], where $n \geq 2$ and $h \geq 0$. The parameter n is the degree (or radix) of the switch-nodes, each of which is connected to α master-nodes and β slave-nodes (so called in [15][2]), and h is the depth of the recursion in the construction of HCN(n, h). In the context of the DCNs HCN and BCN, it is always the case that $n = \alpha + \beta$.

For ease of notation, let G^h (temporarily) denote HCN(n, h) (we suppress the parameter n for convenience). The graph G^0 is the n-star graph, comprising a switch-node adjacent to n server-nodes (of which α are master-nodes and β are slave-nodes). Label the master-nodes $0, 1, \ldots, \alpha - 1$.

For $h \geq 1$, construct the graph G^h from α disjoint copies of G^{h-1}, labelled $G_0^{h-1}, G_1^{h-1}, \ldots, G_{\alpha-1}^{h-1}$. Label any master-node of G^h with the h-tuple[3] $u = u_h u_{h-1} \cdots u_0$, where u_h is the index of the copy of G^{h-1} containing the master-node, and $u_{h-1} \cdots u_0$ is the label of the master-node within $G_{u_h}^{h-1}$ (implicit in the definition of the labels of master-nodes is that $0 \leq u_i < \alpha$, for each $0 \leq i \leq h$). Note how for $0 \leq \gamma < h$, there are various copies of G^γ within G^h so that each of these can be canonically labelled $u_h u_{h-1} \ldots u_{\gamma+1}$ according to its place in the recursive hierarchy.

We use the labels of master-nodes to define the *level h links* that join master-nodes in different disjoint copies of G^{h-1} so as to form G^h. A pair of master-nodes forms a level h link in G^h, where

[1] Strictly speaking, this is a unicast routing algorithm, but we do not discuss any other sort in this paper.

[2] We regret the imagery invoked by the terms 'master' and 'slave'; however, we have elected to retain this terminology from [15] for clarity.

[3] Throughout this paper we write $x_k x_{k-1} \cdots x_0$ to denote the $(k+1)$-tuple $(x_k, x_{k-1}, \ldots, x_0)$.

$h \geq 1$, if, and only if, the labels u and w of these master-nodes are such that

$$u = u_h \underbrace{u'_h u'_h \cdots u'_h}_{h \text{ times}} \text{ and } w = u'_h \underbrace{u_h u_h \cdots u_h}_{h \text{ times}},$$

where $u_h \neq u'_h$. Consequently, for every $1 \leq j \leq h$, there are links joining the master-nodes with labels

$$u_h u_{h-1} \cdots u_{j+1} u_j \underbrace{w_j w_j \cdots w_j}_{j \text{ times}} \text{ and}$$
$$u_h u_{h-1} \cdots u_{j+1} w_j \underbrace{u_j u_j \cdots u_j}_{j \text{ times}},$$

for which $u_j \neq w_j$.

As well as labels for the master-nodes, we also define labels for the switch-nodes and slave-nodes. The switch-node adjacent to the α master-nodes $u_h u_{h-1} \cdots u_1 u_0$, for $0 \leq u_0 < \alpha$, is given the label $u_h u_{h-1} \cdots u_1$, and the β slave-nodes adjacent to this switch-node are labelled $u_h u_{h-1} \cdots u_1 y$, for $\alpha \leq y < \alpha + \beta$. In consequence, the graph G^h has: α master-nodes of degree 1; $\alpha^h \beta$ slave-nodes of degree 1; $\alpha(\alpha^h - 1)$ master-nodes of degree 2; and α^h switch-nodes of degree n. Henceforth, we identify nodes with their labels.

We shall need two alternative identifiers. Let $v = u_h u_{h-1} \cdots u_1 y$ be a slave-node of G^h. We define an identifier for v within G^h as

$$uid_h(v) = \left(\sum_{i=1}^{h} u_i \alpha^{i-1} \right) \beta + (y - \alpha). \qquad (1)$$

The function uid_h is a bijective mapping of the slave-nodes of G^h to the set $\{0, 1, \ldots, \alpha^h \beta - 1\}$. Consider some copy B of G^γ within G^h, where $0 \leq \gamma < h$. We define an identifier for B within G^h as

$$hid_\gamma(B) = \left(\sum_{i=\gamma+1}^{h} u_i \alpha^{i-(\gamma+1)} \right). \qquad (2)$$

The function hid_γ is a bijective mapping of copies of G^γ within G^h to the set $\{0, 1, \ldots, \alpha^{h-\gamma} - 1\}$.

We now revert back to our original notation and refer to G^h as HCN(n, h). The DCN HCN$(8, 2)$ can be visualized as in Fig. 1, where $\alpha = 4$ and $\beta = 4$. The slave-nodes are in white, the master-nodes are in black, and, as we have described above, the label of any master-node is obtained by appending a number from $\{0, 1, 2, 3\}$ to the label of the adjacent switch-node.

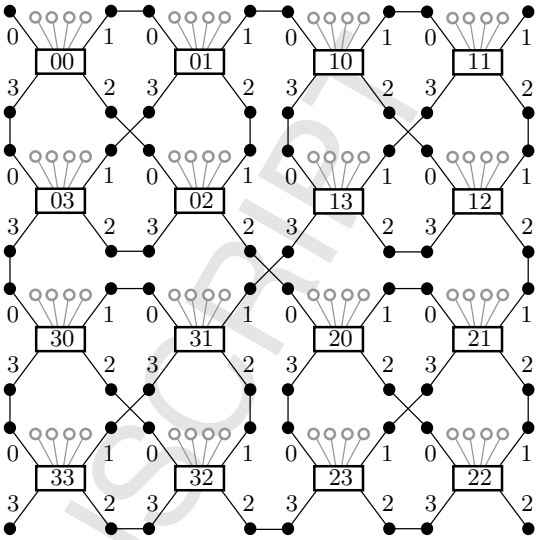


Figure 1: The DCN HCN$(8, 2)$.

Notice that the slave-nodes play no part in the construction of HCN(n, h), besides the fact that there are β of them within each copy of HCN$(n, 0)$. They are, however, used in Section 2.2 when we construct the DCN BCN.

2.1.1. WK-recursive networks

Observe that if the slave-nodes are ignored in HCN(n, h) and each switch-node is replaced with a clique on its adjacent α master-nodes then the resulting graph is isomorphic to the WK-recursive network WK(α, h), first defined in [25]. To our knowledge, this observation is novel (and first mentioned in the preliminary version of our paper [24]).

Replacing the switch-nodes in HCN(n, h) with α-cliques is, in fact, a very natural abstraction for developing routing algorithms in server-centric DCNs: the links used in a route within WK(α, h) correspond to hops in the corresponding route in HCN(n, h). Thus, path-length in WK(α, h) corresponds exactly to hop-length in HCN(n, h).

WK-recursive networks have been extensively studied since they were first defined and, as we shall see later, we can use the analysis of these networks in order to better understand the topological properties of the DCNs HCN and BCN.

Formally, the *WK-recursive network* WK(α, h) is defined so that: it has node-set $\{0, 1, \ldots, \alpha - 1\}^{h+1}$; and there are links

$$(i_h i_{h-1} \ldots i_2 i_1 x, i_h i_{h-1} \ldots i_2 i_1 x'),$$

where $i_1, i_2, \ldots, i_h, x, x' \in \{0, 1, \ldots, \alpha - 1\}$, with $x \neq x'$, as well as links

$$(i_h i_{h-1} \cdots i_{j+1} i_j \underbrace{i_{j'} i_{j'} \cdots i_{j'}}_{j \text{ times}}, \\ i_h i_{h-1} \cdots i_{j+1} i_{j'} \underbrace{i_j i_j \cdots i_j}_{j \text{ times}}),$$

where $0 < j \leq h$ and $i_j \neq i'_j$.

2.2. The DCN BCN

We construct BCN$(\alpha, \beta, h, \gamma)$ (as in Sections 3.2 and 3.3 of [15]) by using slave-nodes to interconnect disjoint copies of HCN(n, h) as explained below. Let $h \geq 0$, $\gamma \geq 0$, $\alpha \geq 2$, and $n = \alpha + \beta$ be given.

Let us outline BCN$(\alpha, \beta, h, \gamma)$ where $h \geq \gamma$, since in the case for which $h < \gamma$, the DCNs BCN$(\alpha, \beta, h, \gamma)$ and HCN(n, h) are defined to be identical. We begin by taking $\alpha^\gamma \beta + 1$ disjoint copies of HCN(n, h). Recall, from Section 2.1, that each copy of HCN(n, h) is composed of $\alpha^{h-\gamma}$ disjoint copies of HCN(n, γ). We now define a perfect matching amongst the slave-nodes of all the copies of HCN(n, γ) so that there is exactly one link joining each such pair of copies of HCN(n, γ). However, we impose restrictions on this matching.

For brevity, let $s = \alpha^\gamma \beta$, let $t = \alpha^{h-\gamma} - 1$, and let $B_0, B_1, \ldots B_s$ be the $s + 1$ disjoint copies of HCN(n, h) within BCN$(\alpha, \beta, h, \gamma)$. Within each B_u, for $0 \leq u \leq s$, let B_u^v be the copy of HCN(n, γ) such that $hid_\gamma(B_u^v) = v$ (recall that $0 \leq v \leq t$). For a slave-node x in B_u^v, define $id(x) = (u, v, uid_\gamma(x))$.

We now add slave-node-to-slave-node links to complete the construction of BCN$(\alpha, \beta, h, \gamma)$. For each fixed v, add links to construct a perfect matching of pairs of slave-nodes in $B_0^v, B_1^v, \ldots, B_s^v$, such that for each i and j, where $i \neq j$, there is exactly one link joining B_i^v with B_j^v. The overall scheme can be visualised in Fig. 2.

There are various different matchings that might be employed and we consider the two that were highlighted in [15], called *slave-connection-rule*-1 and *slave-connection-rule*-2. They are defined as follows. Fix $0 \leq v \leq t$ and let $0 \leq i < j \leq s$.

- **Slave-connection-rule-**1: the sub-networks B_i^v and B_j^v are joined by the link joining the slave-node x for which $id(x) = (i, v, j - 1)$ and the slave-node y for which $id(y) = (j, v, i)$.

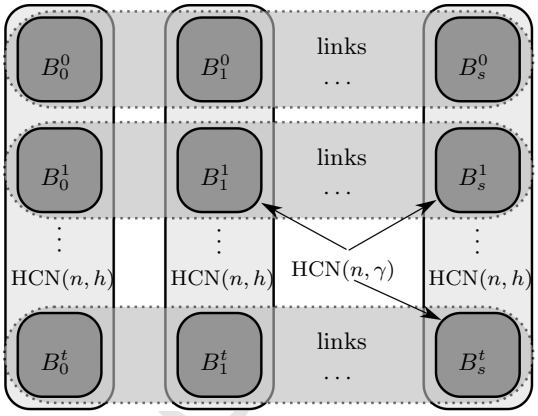


Figure 2: The network BCN$(\alpha, \beta, h, \gamma)$ when $h \geq \gamma$.

- **Slave-connection-rule-**2: the sub-networks B_i^v and B_j^v are joined by the link joining the slave-node x for which $id(x) = (i, v, j - i - 1)$ and the slave-node y for which $id(y) = (j, v, s - j + i)$.

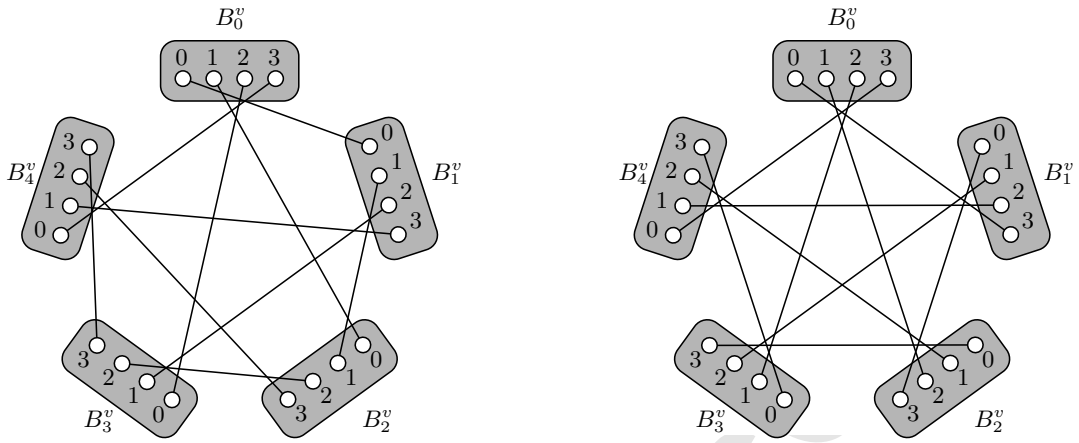
The two connection rules can be visualised in Figs. 3a and 3b, respectively. Slave-connection-rule-1 is essentially identical to the connection rule used to define DCell in [14], and slave-connection-rule-2 is a connection rule derived in [3] but more recently used in Generalized DCell in [17]. Of course, there are many more such matching connection rules that might be considered.

In consequence, the DCN BCN$(\alpha, \beta, h, \gamma)$, for $h \geq \gamma$, has: $(\alpha^\gamma \beta + 1)\alpha$ master-nodes of degree 1; $(\alpha^\gamma \beta + 1)\alpha^h \beta$ slave-nodes of degree 2; $(\alpha^\gamma \beta + 1)\alpha(\alpha^h - 1)$ master-nodes of degree 2; and $(\alpha^\gamma \beta + 1)\alpha^h$ switch-nodes of degree n.

3. One-to-one Routing in the DCN HCN

We first describe the one-to-one routing algorithm for HCN(n, h) called *FdimRouting* that was derived in [15] before describing an improved one-to-one, minimal routing algorithm called *NewFdimRouting*. In essence, the algorithm *FdimRouting* is that obtained in [4, Section 3.1] and the algorithm *NewFdimRouting* is actually that obtained in [4, Section 3.2].

Throughout this paper, for a given routing algorithm R and server-nodes *src* and *dst*, a path from *src* to *dst* computed by R is denoted by R(src, dst), and the hop-length of the path by $|\mathtt{R}(src, dst)|$.



(a) Slave-connection-rule-1. (b) Slave-connection-rule-2.

Figure 3: Slave-node links in BCN$(2, 2, 1, 1)$.

3.1. Routing with FdimRouting

The one-to-one routing algorithm for HCN(n, h) from [15], named *FdimRouting*, proceeds as follows. We first describe the algorithm for master-nodes. Let *src* and *dst* be master-nodes in HCN(n, h), and let γ be the smallest parameter such that *src* and *dst* are contained in the same sub-network HCN(n, γ) of HCN(n, h). If $\gamma = 0$ then *src* and *dst* are connected to the same switch (and they may be identical); otherwise, let B^a and B^b be the distinct copies of HCN$(n, \gamma - 1)$ containing *src* and *dst*, respectively, where a and b are the indices, from the set $\{0, 1, \ldots, \alpha - 1\}$, of B^a and B^b within the copy of HCN(n, γ) that contains them. Let (dst', src') be the link that joins B^a to B^b, with dst' in B^a and src' in B^b. *FdimRouting* builds a path from *src* to *dst* by recursively building a path from *src* to dst' within B^a and from src' to *dst* within B^b, and then joining these two paths by the link (dst', src').

Given *src* and *dst*, the labels of dst' and src' are easily derived from the definition of HCN(n, h). All four of these master-nodes are contained in the same copy of HCN(n, γ), so they must begin with the same prefix, $u_h u_{h-1} \cdots u_{\gamma+2} u_{\gamma+1}$ (the prefix is empty in the degenerate case where $\gamma = h$). The nodes dst' and src' join B^a and B^b, and so we have

$$dst' = u_h u_{h-1} \cdots u_{\gamma+2} u_{\gamma+1} a \underbrace{b b \cdots b}_{\gamma \text{ times}}$$

$$src' = u_h u_{h-1} \cdots u_{\gamma+2} u_{\gamma+1} b \underbrace{a a \cdots a}_{\gamma \text{ times}}.$$

Under our identification of HCN(n, h) (with its slave-nodes removed) with WK(α, h), this algorithm from [15] is actually just the routing algorithm for WK(α, h) from [4, Section 3.1].

It was shown in Theorem 4 in [15] that *FdimRouting* yields a path joining any two master-nodes of HCN(n, h) of length at most $2^{h+1} - 1$. Consequently, the length of a shortest path between any two master-nodes of HCN(n, h) is at most $2^{h+1} - 1$. Whilst this upper bound was noted in [15], it was left unresolved as to whether the length of a shortest path between any two master-nodes of HCN(n, h) is exactly $2^{h+1} - 1$ in the worst-case. However, that this is the case was proven in [4] (we translate from the language of a WK-recursive network to that of a DCN HCN).

Lemma 3.1. *[4, Lemma 2.1] Let $n \geq 2$ and $h \geq 1$. There exist source and destination master-nodes of HCN(n, h) such that the hop-length of a shortest path from the source to the destination is exactly $2^{h+1} - 1$.*

It is trivial to implement the algorithm *FdimRouting* as a source-routing algorithm so that it has $O(h2^h)$ time complexity (and not $O(2^h)$ as was stated in [4, 15]; for even writing the route takes $O(h2^h)$ time where we assume that $n = O(1)$). Also, it is not difficult to see that *FdimRouting* can be implemented as a distributed-routing algorithm so that the time taken for each iterim master-node to compute the next master-node on the route is $O(h)$.

3.2. Routing with NewFdimRouting

As was noted in [4, Section 3.2], the routing algorithm *FdimRouting* is not a minimal routing al-

gorithm and can be improved. Consider applying the routing algorithm *FdimRouting* to the source master-node $(0,1,1)$ and the destination master-node $(2,1,1)$ of HCN$(5,2)$ (where $\alpha = 3$ and $\beta = 2$). The resulting path is:

$$(0,1,1) \to (0,1,2) \to (0,2,1) \to (0,2,2) \to (2,0,0) \to (2,0,1) \to (2,1,0) \to (2,1,1).$$

However, the path

$$(0,1,1) \to (1,0,0) \to (1,0,2) \to (1,2,0) \to (1,2,2) \to (2,1,1)$$

is shorter.

The algorithm given as Algorithm 1, which we call *GetShortest*, was proven in [4, Section 3.2] to yield a minimal routing algorithm for WK(α, h) as we now explain. We think of all routing algorithms for WK(α, h) as also being routing algorithms for HCN(n, h) when the slave-nodes are ignored, and *vice versa*.

Algorithm 1

Require: $u = u_h u_{h-1} \cdots u_0$ and $v = v_h v_{h-1} \cdots v_0$ are distinct nodes in WK(α, h).
  function GetShortest(u, v)
    $P \leftarrow$ *FdimRouting*(u, v)
    $l \leftarrow |P|$
    $i \leftarrow$ largest index such that $u_i \neq v_i$
    for $z \in \{0, 1, \ldots, \alpha - 1\} \setminus \{u_i, v_i\}$ **do**
      $l_z^u \leftarrow |$*FdimRouting*$(u, u_h u_{h-1} \cdots \cdots u_{i+1} u_i \underbrace{zz \cdots z}_{i \text{ times}})|$
      $l_z^v \leftarrow |$*FdimRouting*$(v, v_h v_{h-1} \cdots \cdots v_{i+1} v_i \underbrace{zz \cdots z}_{i \text{ times}})|$
      $l_z = l_z^u + l_z^v + 2^i + 1$
    end for
    if $l \leq \min\{l_z : u_i \neq z \neq v_i\}$ **then**
      return P
    else
      return z, where $l_z = \min\{l_z : u_i \neq z \neq v_i\}$
    end if
  end function

Let the nodes $u = u_h u_{h-1} \cdots u_0$ and $v = v_h v_{h-1} \cdots v_0$ of WK(α, h) be distinct. The algorithm *GetShortest*(u, v) outputs either a path or a value from $\{0, 1, \ldots, \alpha - 1\}$. If *GetShortest*$(u, v)$ outputs the path P then we define the path *NewFdimRouting*(u, v) as P (in this case, the path P is actually the path *FdimRouting*(u, v)). Alternatively, if *GetShortest*(u, v) outputs the value $z \in \{0, 1, \ldots, \alpha - 1\}$ then we define the path *NewFdimRouting*(u, v) as follows. Let

- P_u be the path

  $$\textit{FdimRouting}(u, u_h u_{h-1} \cdots u_{i+1} u_i \underbrace{zz \cdots z}_{i \text{ times}})$$

- Q be the path

  $$\textit{FdimRouting}(u_h u_{h-1} \cdots u_{i+1} z \underbrace{u_i u_i \cdots u_i}_{i \text{ times}}, v_h v_{h-1} \cdots v_{i+1} z \underbrace{v_i v_i \cdots v_i}_{i \text{ times}})$$

- P_v be the path

  $$\textit{FdimRouting}(v_h v_{h-1} \cdots v_{i+1} v_i \underbrace{zz \cdots z}_{i \text{ times}}, v).$$

The path *NewFdimRouting*(u, v) is defined as $P_u + link_1 + Q + link_2 + P_v$, where $link_j$ is the link joining the terminals of the appropriate paths, for $j \in \{1, 2\}$.

It is not the case that *FdimRouting* has to be executed in the for-loop of Algorithm 1 in order to obtain l_u and l_v; for, by [4, Lemma 3.3], the following is true.

Theorem 3.2. *The length of a shortest path joining the nodes* $\underbrace{zz \cdots z}_{h+1 \text{ times}}$ *and* $u_h u_{h-1} \cdots u_0$ *of the WK-recursive network WK*(α, h) *is*

$$\sum_{i = 0, 1, \ldots, h \text{ where } u_i \neq z} 2^i.$$

Consequently, we can calculate the length of a shortest path, along with which route it takes, without computing any actual path; a simple numeric calculation suffices. Once we have this information, we can build the actual path as specified by the output value z.

Let us now return to when at least one of our source and destination nodes in HCN(n, h) is a slave-node (this was left blurred in [15]). W.l.o.g., suppose that our source is a slave-node. We calculate the length of a shortest path between every master-node adjacent to the same switch-node as

the source and: the destination node, if the destination is a master-node; or to every master-node one hop from the destination node, if the destination is a slave-node. We take the resulting path of minimal hop-length as our shortest path. We note that Theorem 3.2 assists significantly with this computation.

As we noted above, *FdimRouting* can be implemented as both a source-routing and a distributed-routing algorithm. This is also true for *NewFdimRouting*. When implemented as a source-routing algorithm, the repeated numeric computations (so as to ascertain the path to take) take $O(h)$ time (recall, n is assumed to be $O(1)$); so, the complexity remains at $O(h2^h)$. When implemented as a distributed-routing algorithm, as well as carrying the source and the destination within the packet header, the value z, output from *GetShortest*, must also be carried. When it is, the time taken for each interim node to compute the next node on the route remains at $O(h)$.

4. One-to-one Routing in the DCN BCN

In this section we describe the routing algorithms for the DCN BCN as derived in [15], followed by a novel, improved routing algorithm.

4.1. Routing with BdimRouting

We first describe the routing algorithm *BdimRouting* from [15]. Let *src* and *dst* be server-nodes in $\mathrm{BCN}(\alpha, \beta, h, \gamma)$, where $h \geq \gamma$ (*src* might be a master-node or a slave-node, as might *dst*). Recall that $\mathrm{BCN}(\alpha, \beta, h, \gamma)$ is composed of disjoint copies of $\mathrm{BCN}(n, \gamma)$, labelled B_u^v, for each $(u, v) \in \{0, 1, \ldots, s\} \times \{0, 1, \ldots, t\}$, as depicted in Fig. 2 (s and t are as defined in Section 2.2). If *src* and *dst* reside in the same copy of $\mathrm{HCN}(n, h)$, then *BdimRouting*(src, dst) returns the path returned by *FdimRouting*(src, dst). Thus, we assume that *src* is in B_u^v and *dst* is in $B_{u'}^{v'}$, for some $u \neq u'$ (note that we might have that $v = v'$).

Let (x, x') be the unique slave-node-to-slave-node link joining B_u^v to $B_{u'}^v$ (see Section 2.2). The routing algorithm *BdimRouting* returns the path *FdimRouting*$(src, x) + (x, x') +$ *FdimRouting* (x', dst). Alternatively, (x, x') can be the unique link joining $B_u^{v'}$ to $B_{u'}^{v'}$; note that if $v = v'$ then the two alternatives produce identical paths.

Of course, we can immediately improve *BdimRouting* by using the algorithm *NewFdimRouting* instead of the algorithm *FdimRouting*; however, irrespective of whether we use *FdimRouting* or *NewFdimRouting*, the algorithms outlined above do not necessarily yield shortest paths within $\mathrm{BCN}(\alpha, \beta, h, \gamma)$. A concrete example as to why this is the case is given in [24], but we shall now move forward with a more sophisticated improvement of *BdimRouting*.

4.2. Routing with NewBdimRouting

The routing algorithm *BdimRouting* from [15], employed within $\mathrm{BCN}(\alpha, \beta, h, \gamma)$, where $h \geq \gamma$, and outlined above, is such that if the source is in B_u and the destination is in $B_{u'}$, where $u \neq u'$, then the route derived remains entirely within B_u and $B_{u'}$. *NewBdimRouting*, described as Algorithm 2, performs an intelligent search to find a *proxy* copy of $\mathrm{HCN}(n, h)$, $B_{u''}$, through which a shorter path can be routed. We discuss the intelligent construction of the set of proxies at the end of this section, so that this search feature appears merely in the form of the set *Proxies* in Algorithm 2; for the moment, think of *Proxies* as including all possibilities for u'', of which there are $\alpha^\gamma \beta - 1$.

Algorithm 2 The routing algorithm *NewBdimRouting* in $\mathrm{BCN}(\alpha, \beta, h, \gamma)$, where $h \geq \gamma$. Note that we may have $v = v'$ (below). The call to *BdimRouting*(src, dst) employs *NewFdimRouting* in place of *FdimRouting*.

Require: *src* and *dst* are server-nodes in B_u^v and $B_{u'}^{v'}$, respectively.
  function NEWBDIMROUTING(src, dst)
    if $u = u'$ **then**
      return *NewFdimRouting*(src, dst)
    end if
    $Q \leftarrow$ *BdimRouting*(src, dst)
    for $u'' \in$ *Proxies* **do**
      $(x, x'') \leftarrow$ link joining B_u^v to $B_{u''}^v$
      $(y'', y) \leftarrow$ link joining $B_{u''}^{v'}$ to $B_{u'}^{v'}$
      $P_{u''} \leftarrow$ *NewFdimRouting*(src, x)
        $+(x, x'')$ + *NewFdimRouting*(x'', y'')
        $+(y'', y)$ + *NewFdimRouting*(y, dst)
    end for
    return a path of shortest hop-length in
      $\{P_{u''} : u'' \in \{0, 1, \ldots, s\} \setminus \{u, u'\}\} \cup \{Q\}$
  end function

Of course, Theorem 3.2 makes the implementation of *NewBdimRouting* trivial, so that the search

for an optimal choice of $B_{u''}$ to route through involves examining the hop-length of exactly one path for each candidate proxy in *Proxies*. When implemented as a source-routing algorithm, and given our comments earlier as regards the implementation of *NewFdimRouting*, once $B_{u''}$ is found, *NewBdimRouting* constructs that chosen path in $O(h2^h)$ steps; for it is essentially 3 repetitions of *NewFdimRouting*. As regards the implementation of *NewBdimRouting* as a distributed-routing algorithm, again the time complexity is $O(h)$. However, the packet header must also carry the 3 different z's corresponding to the 3 executions of *NewFdimRouting*, as well as a parameter detailing which $B_{u''}$ *NewBdimRouting* transits through.

4.3. Selecting proxies and alternative routing

An exhaustive search through candidate proxies $B_{u''}$, for each u'' in $\{0, 1, \ldots, s\} \setminus \{u, u'\}$, together with the associated overheads of this search, can be avoided, albeit with a potential loss of path quality. We construct the set *Proxies*, used in Algorithm 2, following methods analogous to those developed in [7]. In short, for the recursively-defined DCNs considered in [7], namely (Generalized) DCell and FiConn, only proxies 'within a small radius' of *src* or *dst* are considered, where 'within a small radius' is interpreted as within some recursive copy with the recursion parameterized by the radius. For us, this locality translates as follows. Fix $\gamma \geq r \geq 0$ and let B_u (resp. $B_{u'}$) be the copy of HCN(n, h) within BCN$(\alpha, \beta, h, \gamma)$ in which *src* (resp. *dst*) lies. Moreover, let $B_{u,r}$ (resp. $B_{u',r}$) be the copy of HCN(n, r) within B_u (resp. $B_{u'}$) in which *src* (resp. *dst*) lies. Define the set $A_r = \{u'' : 0 \leq u'' \leq s, u' \neq u'' \neq u,$ there is a link from a slave-node of $B_{u,r}$ to a slave-node of $B_{u''}\}$, with the set A'_r defined analogously but w.r.t. *dst* and $B_{u',r}$ as opposed to *src* and $B_{u,r}$. Set $Proxies = A_r \cup A'_r$. We shall call the version of *NewBdimRouting* that constructs *Proxies* in this way *NewBdimRouting*$_r$, where r is the 'radius' parameter. Clearly, if $r = \gamma$ then we obtain the exhaustive search. Consequently, we can think of *NewBdimRouting* as being a suite of routing algorithms, with one algorithm for each $0 \leq r \leq \gamma$.

In choosing a value for r, above, there is clearly a trade-off between the breadth of search and the overheads associated with undertaking this search: the wider the search, the more chance of finding a shorter path, but the longer this search takes to undertake. In [7], it was found that restricting searches to 'small radius' led to significant performance gains yet did not unduly compromise the quality of the resulting paths found. Consequently, buoyed by the results of [7], we have proceeded analogously here. As such, we compare *NewBdimRouting*$_\gamma$ and *NewBdimRouting*$_1$ in our experimental analysis.

The (suite of) routing algorithm(s) *NewBdimRouting* serves as a prototype for a class of routing algorithms that search for shortest paths. With the notation of Algorithm 2, at present we route: from *src* in B_u^v to $B_{u''}^v$, via a slave-node-to-slave-node link; then on through $B_{u''}$ to $B_{u''}^{v'}$; and finally, via a slave-node-to-slave-node link, from $B_{u''}^{v'}$ to *dst* in $B_{u'}^{v'}$. Alternatively, we might route: from *src* in B_u^v to $B_u^{v''}$, for some v''; then via a slave-node-to-slave-node link to $B_{u''}^{v''}$, for some u''; then on through $B_{u''}$ to $B_{u''}^{v'}$; and finally, via a slave-node-to-slave-node link, to *dst* in $B_{u'}^{v'}$. In short, there are other as yet unexplored 'dimensions' within which to search for shorter paths. Of course, more searching leads to additional computational overheads. Another alternative is to build paths that pass through not just B_u, $B_{u''}$, and $B_{u'}$, for some u'', but through B_u, $B_{u''}$, $B_{u'''}$, and $B_{u'}$, for some u'' and u''', in the search for yet shorter paths. Of course, this increases the searching overheads given that not only do potential values for u'' have to be explored but potential values for u''' also. We leave the search for efficiently implementable improvements to the approach we take here as a direction for future research and return to this comment in our conclusions.

5. Methodology

We undertake an exhaustive empirical analysis of our newly proposed routing algorithm *NewBdimRouting*$_r$, for specific values of r, by comparing its performance with that of *BdimRouting* in BCN$(\alpha, \beta, h, \gamma)$, over a range of parameter values, for a range of traffic patterns and workloads, and with regard to a comprehensive set of metrics. Our metrics cover hop-length, network throughput, and latency, as well as the overall completion time of various workloads.

Our topologies come in two batches. In Batch A, we include the DCNs BCN$(\alpha, \beta, h, \gamma)$, for $(\alpha, \beta) \in \{(7, 2), (6, 3), (5, 4), (4, 5), (3, 6), (2, 7)\}$ and for $h \in \{3, 4\}$, where $1 \leq \gamma \leq h$ (we detail results primarily for $h = 3$ as trends are replicated for $h = 4$

and the resulting sizes of some of the DCNs when $h = 4$ are too big to be practically relevant). We are guided here by the empirical analysis undertaken in [24] where these parameters are chosen so that we might observe performance trends as the parameter values involved gradually increase or decrease. We obtain a wide range of DCNs in terms of the number of server-nodes; for example, even when h is fixed at 3, BCN$(2, 7, 3, 1)$ has $1,080$ server-nodes whereas BCN$(6, 3, 3, 3)$ has $1,261,656$ server-nodes. We also study both slave-connection rules defined earlier, and we use 1-BCN and 2-BCN, respectively, to specify a particular connection rule (with BCN used when we have no need to specify the actual slave-connection rule used). Although we focus on BCN (for $h \geq \gamma$) in this paper, we have indeed also verified the experiments from [24], comparing *NewFdimRouting* to *FdimRouting* in the topologies HCN$(9, h)$, for $3 \leq h \leq 8$, using our new, independently-developed software tool (see Section 5.1).

In our second batch of topologies, Batch B, we experiment with more realistic topologies in that we reflect readily available denominations of switch-ports, namely 24 and 32. The topologies within this second batch are BCN$(3, 21, 3, \gamma)$ and BCN$(3, 29, 3, \gamma)$, for $1 \leq \gamma \leq 3$, as well as BCN$(12, 12, 2, 1)$ and BCN$(12, 12, 3, 0)$. While there are many combinations of parameters that yield networks of realistic size and switch-radix, we are guided by the results of our experiments on the topologies of Batch A and consequently choose higher values of β and lower values of h for the topologies within the second batch. Again, we get a good spread of DCNs in terms of the number of server-nodes, with these numbers ranging from $41,472$ to $677,376$.

We now describe our software tool and the details of our experiments.

5.1. Software tool: `INRFlow`

We conduct our experiments with the open-source software tool *Interconnection Networks Research Flow Evaluation Framework* (`INRFlow`) [8]. `INRFlow` is a *flow-level* network simulation framework for analysing network topologies and routing algorithms under various traffic patterns, workloads, and fault-conditions. For us, a *traffic pattern* is a set of pairs of source and destination nodes, whereas we think of a *workload* as consisting of a set of flows, possibly with some temporal causality imposed on these flows, with a *flow* being a source-destination pair together with a bandwidth reflecting the amount of data intended to be transported from the source to the destination. We often simply refer to traffic patterns as workloads; that is, we think of source-destination pairs as unitary flows and so that there are no temporal causalities between the flows. `INRFlow` constructs the network topology and workload at runtime, routes the flows specified by the workload, using the specified routing algorithm, and, finally, reports statistics.

`INRFlow` has a *static* and a *dynamic* mode. In static mode, flows are routed simultaneously and a link's capacity is assumed to be shared equally among all the flows routed through it. Static mode can handle very large networks and serves to report on raw performance metrics where the causal relationships between flows are not important, such as the mean hop-length of a routing algorithm or preliminary estimations of the throughput. This is the mode commonly found in experimental work within the server-centric DCN literature. However, the static mode does not always accurately reflect network traffic due to its lack of temporal modelling. For this reason, we extend our experimental work by simulating in dynamic mode. In dynamic mode, the links of the network have capacities and each flow is specified with a bandwidth reflecting the data that must be routed. In addition, the workloads might prescribe causal relationships between flows, so that some flows must finish before others begin. Dynamic mode provides a more realistic, flow-level simulation of general real-world workloads, as well as a good estimation of the completion times of a collection of application-inspired workloads.

As we mentioned earlier, the conclusions of our preliminary experiments in [24] are broadly verified; `INRFlow` was developed independently from the purpose-built tool used in [24]. Note that the tool in [24] does not store the topology in memory, unlike `INRFlow`, so we are unable to reproduce specific experiments on very large networks; however, we do reproduce the overall trends observed in [24].

5.2. Traffic Patterns and Workloads

In order to ascertain the performance of our routing algorithms under various conditions, we experiment with a variety of traffic patterns and workloads, most of which are common in the literature and representative of traffic scenarios arising from scale-out, tenanted, cloud-oriented datacenters (as well as smaller, private datacenters), run-

ning a number of simultaneous data-intensive applications, such as Hadoop (see, *e.g.*, [26]) or Spark (see, *e.g.*, [30]).

We use `INRFlow`'s static engine to measure statistics related to path (hop-)length, network throughput, and latency, under the following wide variety of traffic patterns and workloads.

All-to-one: A destination server *dst* is chosen, uniformly at random, and every server sends a flow to *dst*.

Bisection: The network is split uniformly at random into two halves and every server in each half sends a flow to every server in the other half (see [29]).

Butterfly: Every server sends a flow only to a small subset of servers, as opposed to sending to all of them as in a full all-to-all communication. In more detail, the servers are arbitrarily numbered $1, 2, \ldots, N$, and for any $1 \leq k \leq \lfloor \log(N-1) \rfloor$, servers are paired together as follows: the servers are split into batches of $2k$ contiguously-named servers; and within the mth batch, server $2(m-1)k + i$ is paired with server $2(m-1)k + k + i$, for $1 \leq i \leq k$. Every server sends a flow to every one of the (at most) $\lfloor \log(N-1) \rfloor$ servers it has been paired with. See [20] for more details. The butterfly pattern represents an optimised, binary implementation of collective operations.

Hot-region: One million flows are generated so that each source server is selected uniformly at random and where each destination server is chosen according to a hot-region pattern, whereby $\frac{1}{4}$ of the traffic (on average) goes to $\frac{1}{8}$ of the network, with the rest uniform. The hot-region is chosen by arbitrarily naming the servers $1, 2, \ldots, N$ and taking the hot-region to be the servers $1, 2, \ldots, \lfloor \frac{N}{8} \rfloor$.

Many-all-to-all: For a given size s, the network is partitioned uniformly at random into $g = \lceil N/s \rceil$ groups of servers, each of size at most s. Each server sends a flow to all other servers in its group. In this paper we take $s = 1,000$.

Uniform-random: One million flows are generated in which both the source and the destination are chosen uniformly at random.

After assessing the raw performance improvements achieved by our routing algorithms, in terms of hop-length, throughput, and latency, we evaluate how these raw performance improvements translate to more realistic scenarios using `INRFlow`'s dynamic engine and a collection of new, application-inspired workloads. These new workloads cover more representative and realistic network traffic scenarios that can be found in existing datacenters. Note that this is beyond what is normally undertaken as current practice within the server-centric community. Our new workloads, along with a brief justification, can be described as follows.

MapReduce: Our MapReduce workflow is such that we partition the servers into equal-sized groups so that every server is allocated to a group; this partitioning is undertaken uniformly at random. Within each group, a root server, chosen uniformly at random, undertakes a broadcast to the group, so as to partition the original data amongst all servers. Once a server has received its data from the root, it performs the 'mapping' of the data and 'shuffles' it to the other servers via a one-to-all group broadcast. Once a server has received all 'mapped' flows from the other servers in its group, it 'reduces' its data and sends its results back to the group root. The completion time of the MapReduce workflow is measured as the time required to complete all the communications in all of the groups.

MapReduce is the main application model used in the context of datacenter systems for big data analytics (see, *e.g.*, [6]). Were we to work only with one group consisting of all of the servers, MapReduce would be computationally infeasible; consequently, we partition the servers into groups of $1,000$ servers. It is common practice in datacenters where storage is distributed across subsets of servers to partition the servers into groups uniformly at random, so as to reduce the effects of correlated server failures (see, *e.g.*, [23]).

Stencil and sweep: In these workloads, we assume that there is a virtual topology imposed upon the servers, in the form of a d-dimensional grid (this virtual grid is imposed on the servers arbitrarily). Each server sends data to its neighbours in this virtual topology. We illustrate stencil and sweep for $d = 2$ but the general case is analogous. In the sweep

workflow, the corner server $(0, 0)$ sends to its neighbours with all other servers waiting until they have received data from *all* their 'lower order (left and above)' neighbours before sending data to their 'higher order (right and below)' neighbours (the *wavefront* can be visualized as progressing diagonally through the grid from the top-left). In the stencil workflow, all servers send to their neighbours and wait to receive data from *all* of their neighbours. This constitutes a round. When a server has received data from each of its neighbours it can embark on the next round.

Many scientific and engineering parallel applications operate over huge d-dimensional matrices; consequently, we can arrange things so that each server deals with a small sub-matrix of the whole. This partitioning yields a good locality of communication by requiring that servers need only communicate with those servers that are neighbouring in the inherited virtual grid topology. In our workloads, we assume that a single application is using the whole of the datacenter, and for the purposes of this paper, we consider $d = 2, 3$. We varied the number of rounds in the stencil workflow but found that this did not affect the results. Additional details as regards stencil and sweep can be found in [20].

Unstructured applications: We consider workloads following the uniform-random and hot-region patterns, described above, but where we have enhanced causality. We generate flows as prescribed in each workload; however, we divide the flows into phases, uniformly at random, so that each phase has a fixed number of flows. We experimented with the number of flows in each phase being $1,000$, $10,000$, $100,000$, and 1 million, so that the total number of flows is always 10 million. Each phase requires all the flows from the previous phase to be delivered before it can begin. The smaller the phase size, the more tightly-coupled the application, *i.e.*, the higher the causality.

Unstructured workloads often arise when considering system management traffic, system schedulers based on work-stealing (see, *e.g.*, [22]), or graph analytics applications (a key application area within datacenters; see, *e.g.*, [16]).

Note that there are numerous other traffic patterns and workloads that we might consider such as patterns relating to multicasting or broadcasting. However, our chosen traffic patterns and workloads are representative and cover many others. For example, if one considers one-to-many then one sees that it is embedded within uniform-random and hot-region as because of the large number of flows that we generate, a single source is likely to have a number of associated destinations. Also, one-to-many is naturally embedded within MapReduce. There is nothing really to be gained by extending our chosen range of traffic patterns and workloads.

5.3. Hop-length experiments

We undertake three types of hop-length experiment, along with a study of the efficiency of proxy selection in Algorithm 2. Our focus is primarily on the uniform-random traffic pattern but we also look at some of the other traffic patterns defined above.

First, we adopt the uniform-random traffic pattern and compare the mean hop-lengths of four routing algorithms for BCN$(\alpha, \beta, h, \gamma)$, namely: the newly introduced algorithms *NewBdimRouting*$_1$ and *NewBdimRouting*$_\gamma$; the previously known algorithm *BdimRouting*; and a breadth-first search algorithm (*BFS*). Moreover, we do this for both of our slave-connection rules. Our algorithm *BFS* provides a benchmark (note that although we can provide shortest paths via a brute-force application of our algorithm *BFS*, purely for statistical purposes, no efficient shortest-path routing algorithm is known for the DCN BCN; moreover, the implementation of a *BFS* as a DCN routing algorithm is computationally infeasible). We experiment with DCNs from Batch A and Batch B.

In our second hop-length experiment, we stay with the uniform-random traffic pattern and look at the distribution of hop-lengths for the routing algorithms *NewBdimRouting*$_1$ and *BdimRouting* in BCN$(3, 21, 3, 3)$ (from Batch B) with the slave-connection-1 rule, again against the benchmark provided by *BFS*. We choose *NewBdimRouting*$_1$ due to its very good performance against *NewBdimRouting*$_\gamma$ in our first batch of experiments (there is an obvious reduction in implementation overheads too), and the slave-connection-2 rule, given our initial success in comparison with the slave-connection-1 rule.

In our third hop-length experiment, for each of the traffic patterns all-to-one, bisection, butterfly, hot-region, many-all-to-all, and uniform-random, we compare *BdimRouting* and *NewBdimRouting*$_1$ in DCNs selected from Batch A and Batch B, with

respect to the percentage savings made on average as regards the hop-lengths of the paths generated by the two algorithms.

Finally, moving away from explicit hop-lengths, we also consider how often $NewBdimRouting_1$ and $NewBdimRouting_\gamma$ find a shorter path by routing through a proxy $B_{u''}$ instead of going directly from B_u to $B_{u'}$, as described in Algorithm 2. We do this for DCNs selected from Batch A and Batch B.

We say more about our experimental configurations when we evaluate our hop-length experiments in Section 6.1.

5.4. Throughput experiments

Many datacenter applications rely on frequent, data-heavy communications through the network, which puts network throughput at the forefront of performance requirements. We measure throughput via two metrics, one of which is a generalization of the aggregate bottleneck throughput, introduced in [13]. In [13], the *aggregate bottleneck throughput* (ABT) is defined (only) for the all-to-all traffic pattern as the total number of flows multiplied by the throughput of a bottleneck flow, where a *bottleneck flow* is a flow that receives the smallest throughput. However, it is not entirely clear as to the exact intentions behind the ABT definition. For example, calculations in [13, 19] are undertaken not according to the loads on links in the paths underpinning flows according to some specific routing algorithm but: according to the average hop-length of paths and via an appeal to symmetry within the DCN in [13] (here, the DCN is DCell); and according to 'theoretical' shortest-path routing algorithms in [19] ('theoretical' in the sense that calculations, in the DCNs DCell and BCube, are undertaken by graph-theoretic simulations of some shortest path routing algorithms; indeed, an optimal and efficient shortest-path routing algorithm for DCell is as yet unknown). Moreover, the ABT is geared entirely towards all-to-all workloads, whereas we wish to examine different routing algorithms as regards throughput with regards to alternative workloads.

Given the above discussion, we adapt the ABT so that it better suits our purpose. Our generalization of ABT to arbitrary traffic patterns, which (for distinction) we call the *aggregate restricted throughput* (ART), is defined as Fb/f_{bot}, where F is the number of flows in a given traffic pattern, b is the bandwidth of a link, and f_{bot} is the number of flows that are routed through the bottleneck link (it is assumed that flows are shared over any link evenly and that every flow carries the same load). Intuitively, the ART measures the throughput when all flows are routed at the speed of the (slowest) bottleneck flow; this simulates applications that are tightly coupled with flows and which must wait for the completion of all flows.

We introduce here the *aggregate unrestricted throughput* (AUT) (similar to the metric LFTI, proposed in [28]) which is defined as Fb/f_{ave}, where f_{ave} is the average number of flows in each link. Intuitively, the AUT measures the throughput in applications that are loosely coupled with flows, where each flow can be processed as it arrives. Note that, for us, in both ART and AUT, the bottleneck flow is with respect to the actual routing algorithm employed, rather than *BFS* or an analysis undertaken with average hop-lengths and appealing to symmetry within the DCN.

Our throughput experiments focus on the topologies given in Table 1 and the six initial traffic patterns given in Section 5.2. Our hop-length experiments show that proxy routing offers the strongest performance gains for high values of β, as well as high values of γ. Our goal, following these observations, is to evaluate such parameters more deeply, and thus our narrower selection of topologies in these experiments is so guided.

BCN $(\alpha, \beta, h, \gamma)$	server-nodes	switch-nodes	links
(2,7,3,3)*	4,104	456	12,198
(2,7,4,4)	16,272	1,808	48,590
(3,21,3,3)	368,064	15,336	1,102,488
(3,29,3,3)	677,376	21,168	2,029,776
(3,6,3,3)*	39,609	4,401	118,338
(3,6,4,4)	355,023	39,447	1,063,608
(4,5,3,3)	184,896	20,544	553,404
(5,4,3,3)	563,625	62,625	1,688,370
(6,3,3,3)	1,261,656	140,184	3,781,074

Table 1: The DCNs BCN$(\alpha, \beta, h, \gamma)$ considered in our throughput and completion-time experiments. Dynamic experiments focus only on those marked '*'.

5.5. Latency experiments

While datacenters tend to be used as stream-processing systems, and so are typically more susceptible to throughput variations, there are also many datacenter applications which are more sensitive to latency; these include real-time operations or applications with tight user interactions such as

real-time game platforms, on-line sales platforms, and search engines.

For this reason, we also look at the end-to-end latency for *BdimRouting* and $NewBdimRouting_1$ (just as with our hop-length experiments, we work with the uniform-random pattern). We base our analysis on the latencies imposed by the different steps of the communication: the *protocol stack latency*; the *propagation latency*; the *data transmission latency*; and the *routing latency* at the servers. We measure the latency introduced by each of these steps and model the average zero-load latency by considering each step in conjunction with the average hop-length between the servers.

All of the *transmission-latencies*, *i.e.*, protocol stack, propagation, and data, are measured using the standard UNIX `ping` utility, whereas the routing latency is measured within `INRFlow`. These measurements are carried out independently under low load conditions in the same server, a 32-core AMD Opteron 6220 with 256GB of RAM and running Ubuntu 14.04.1 SMP OS. The server and its neighbour are located in the same rack and are connected with short ($<$ 1 mtr.) electrical wires to a 24-port 1Gbit Ethernet switch. This platform is used because it is a good representative of COTS hardware. In this configuration, we actually measure lower bounds on transmission latencies, since we do not consider other instrumentation needed for a server-centric architecture over and above short wires and protocol stack latency. We measure the routing latency with `INRFlow` in all the selected topologies in Table 1.

Note that routing time measured with `INRFlow` provides a conservative estimate of routing latency that benefits *BdimRouting* and penalises $NewBdimRouting_1$. In a real-world implementation of $NewBdimRouting_1$, where latency is truly critical, a number of optimisations could be applied that would reduce the overheads of $NewBdimRouting_1$ relative to those of *BdimRouting*; for example, using a cache of recent destinations and proxies at each server-node, or even full table look-ups. Thus, since our measured routing latency is an upper bound, and our measured transmission latency is a lower bound, the real proportion of routing latency to transmission latency would be smaller than it is in our measurements. Consequently, hop-length reduction will have a greater impact on the overall latency.

5.6. Completion-time experiments

Our primary objective is to design routing algorithms that reduce the overall execution time of application-like workloads. This requires a more sophisticated modelling in which flows are generated and consumed according to realistic application operation and the maintenance of causal relationships between them. Given that dynamic execution is much more computationally intensive, we restricted our analysis to a few topologies only (marked with a '*' in Table 1), but the consistency of the results with those of the other experiments described in this section suggests that dynamic experiments in other topologies will yield similar results. In order to give some real scale and motivated by the capability of many low-cost COTS hardware components, we use flows of size 1Gb and (uniform) link bandwidths of 1Gbps.

6. Experimental evaluation

We now give an evaluation of the experimental results we obtained when we undertook the experiments laid out in Sections 5.3 to 5.6.

6.1. Hop-length evaluation

As regards our first hop-length experiment, the bar charts in Figs. 4a to 4d detail the percentage hop-length savings of the different versions of *NewBdimRouting*, benchmarked against *BFS*, over *BdimRouting* (note that in Fig. 4a, $NewBdimRouting_1$ and $NewBdimRouting_\gamma$ are one and the same; note also that in the legend for Figs. 4a to 4d, and elsewhere, we use the abbreviation *nB* for *NewBdimRouting*). With reference to Section 5.3, the charts in Figs. 4a to 4d result from the generation of 1 million uniform-random flows. Both $NewBdimRouting_\gamma$ and $NewBdimRouting_1$ yield hop-length gains when compared with *BdimRouting*, and $NewBdimRouting_1$ performs almost as well as $NewBdimRouting_\gamma$ in spite of not undertaking as extensive a search for proxies. Our experiments also confirm the trends observed in [24] that for a fixed switch-node radix n, the hop-length savings decrease marginally with decreasing β; in addition, as γ approaches h, the savings are much more pronounced. Perhaps surprisingly, the slave-connection rule also has a significant effect: both $NewBdimRouting_\gamma$ and $NewBdimRouting_1$ make far greater gains in 2-BCN$(\alpha, \beta, h, \gamma)$ than they do

in 1-BCN(α, β, h, γ). For example, the percentage gain of *NewBdimRouting*$_1$ over *BdimRouting* in 1-BCN$(3, 6, 3, 3)$ is just over 14%, whereas it is around 26% in 2-BCN$(3, 6, 3, 3)$.

The high performance gains for $\gamma = h$ are tempered somewhat by weak performance gains when γ is small in comparison with h. This latter remark can be seen to apply for both smaller and larger DCNs; for example, with BCN$(6, 3, 3, 1)$, which has $36,936$ server-nodes, and with BCN$(12, 12, 2, 1)$, which has $501,120$ server-nodes (the improvement is less than 3% for both slave-connection methods). The reason for this weakness is evidenced in the plots for HCN in [24], which show that, in spite of the proven (hop-length) optimality of *NewFdimRouting*, the gains within HCN(n, h) are not large. Therefore, the majority of the improvement that *NewBdimRouting* has to offer is gained by making a strategic choice of $B_{u''}$, which results in three paths within copies of HCN(n, h) that are shorter than the two paths in the copies of HCN(n, h) that are employed by *BdimRouting*. When γ is small, the number of copies of HCN(n, h) in BCN$(\alpha, \beta, h, \gamma)$ (namely $\beta\alpha^{\gamma} + 1$) is lessened so that there are fewer choices for $B_{u''}$ (see Algorithm 2); furthermore, the potential for hop-length savings using *NewBdimRouting* is inherently limited because if *src* is in B_u^v and *dst* is in $B_{u'}^{v'}$, the distance from $B_{u''}^v$ to $B_{u''}^{v'}$ within any copy $B_{u''}$ of HCN(n, h) needs to be covered regardless of the choice of $B_{u''}$. The above also explains the degrading performance for fixed radix n and decreasing β: as β decreases, so do the number of choices for $B_{u''}$. As a result, there is less potential for reductions in hop-length to be gained this way.

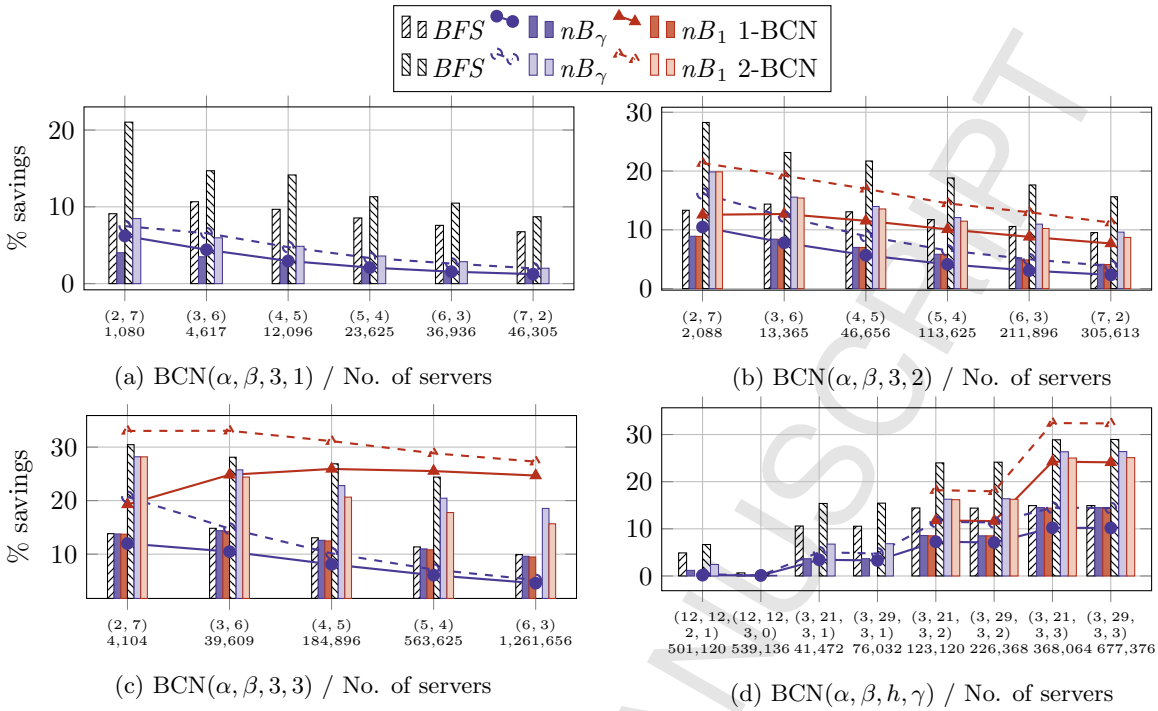
However, it appears that there is potential for strong gains even when γ is small, evidenced by the performance of *BFS* plotted in Figs. 4a and 4b. We discuss alternative routing algorithms in Sections 4.3 and 7, with the caveat that they may incur too much search overhead to be efficient.

As regards our second hop-length experiment, the plots in Figs. 5a and 5b are bar charts showing the normalized distribution of hop-lengths of the paths that were routed using *BdimRouting*, *BFS*, and *NewBdimRouting*$_1$ in 1-BCN$(3, 21, 3, 3)$ and 2-BCN$(3, 21, 3, 3)$, respectively (in the legend, and elsewhere, we use the abbreviation *B* for *BdimRouting*). Again, with reference to Section 5.3, the charts in Figs. 5a and 5b result from the generation of 1 million uniform-random flows. We present our results only for BCN$(3, 21, 3, 3)$ as it is a practically feasible DCN (the total number of server-nodes is 368,064 and it can be implemented with 24-port switches); in any case, we found that the trend of our results is replicated for other DCNs. We choose the routing algorithm *NewBdimRouting*$_1$ as it is a more practical version of *NewBdimRouting*$_\gamma$ and performed almost as well as *NewBdimRouting*$_\gamma$ in our first experiment. As expected, bar charts for *NewBdimRouting*$_1$ are skewed to the left, but notice the long tails. This shows that even when *NewBdimRouting*$_1$ makes some of the greatest gains, there are still long paths that are not shortened.

Notice that even-length paths occur much more frequently than odd-length paths. Paths in BCN alternate between hops that pass through switch-nodes and hops that do not, so the parity of the hop-length of a path is dependant upon whether neither, both, or exactly one of its terminal hops includes a switch-node. The data show that having exactly one terminal hop include a switch-node is unlikely in all three of the routing algorithms plotted.

As regards our third hop-length experiment, we use topologies selected from Batch A and Batch B, with the slave-connection-rule-2 (given its success against the slave-connection-rule-1), and additional traffic patterns, as per Section 5.2. In the results in Fig. 6, we can see that the hop-length reduction when using *NewBdimRouting*$_1$ instead of *BdimRouting* is by at least 15%, but can be as high as 42%. On average, we see a little over 25% savings. It is worth noting that the results obtained with the different traffic patterns are rather consistent regardless of the actual pattern. The only exception is butterfly where the improvement seems to be much better than in the others; all-to-one also presents greater variability but not as much as butterfly. This shows that the hop-length improvements obtained by using *NewBdimRouting*$_1$ are maintained across a wide variety of traffic patterns.

Finally, as regards our study of how common 'good' proxies are, the lines plotted in Figs. 4a to 4d detail the percentages of $B_{u''}$s (from *Proxies* in Algorithm 2) which yield a path for the respective version of *NewBdimRouting*$_r$ that is shorter than the one obtained by *BdimRouting*. These plots tell us that a higher concentration of good choices of $B_{u''}$ are reachable from within the copy of HCN$(n, 1)$ in B_u containing *src* or within the copy of HCN$(n, 1)$ in $B_{u'}$ containing *dst* for *NewB-*



(a) BCN($\alpha, \beta, 3, 1$) / No. of servers

(b) BCN($\alpha, \beta, 3, 2$) / No. of servers

(c) BCN($\alpha, \beta, 3, 3$) / No. of servers

(d) BCN(α, β, h, γ) / No. of servers

Figure 4: Percentage hop-length savings of *NewBdimRouting* and *BFS* (bars) over *BdimRouting*, and the mean percentage of $B_{u''}$s (lines) which yield a path for *NewBdimRouting* that is shorter than the one computed by *BdimRouting*.

dimRouting$_1$ than for *NewBdimRouting*$_\gamma$; that is, 'good' proxies are more heavily concentrated within a small radius. For example, Fig. 4c shows that for 2-BCN($\alpha, \beta, 3, 3$) around 30% of $B_{u''}$s yield gains to *NewBdimRouting*$_1$ over *BdimRouting*, yet for *NewBdimRouting*$_\gamma$ (*i.e.*, an exhaustive search of the copies of HCN(n, γ)) that number is as low as 5%. As we have already noted, this reduction in the search space comes at only a very small cost in hop-length savings, since the gains of *NewBdimRouting*$_1$ are generally quite similar to those of *NewBdimRouting*$_\gamma$.

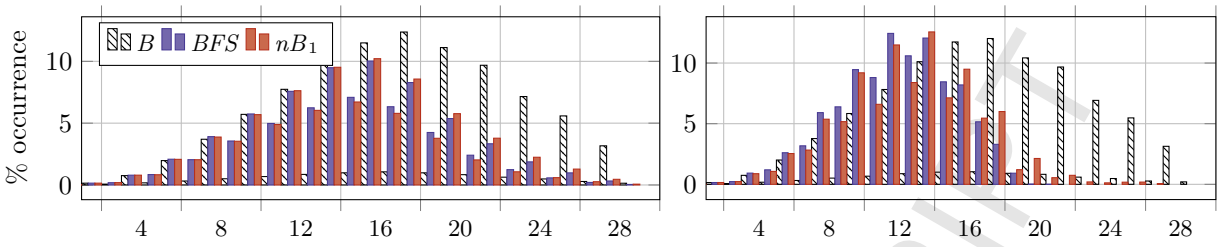
6.2. Throughput evaluation

The hop-length savings on their own provide sufficient motivation to use *NewBdimRouting*$_1$ as they will lead to substantial savings in terms of network utilization and, in turn, energy consumption. We move now to evaluate how the network throughput is affected when using *NewBdimRouting*$_1$. Fig. 7 shows that *NewBdimRouting*$_1$ consistently yields higher AUT than *BdimRouting*, by at least 17%, by an average of 36%, and by up to 72% (the slave-connection-rule-2 is used). Applications that are not tightly coupled with data-communications benefit the most from such a performance gain.

The performance improvements for ART are more volatile, but they are very good for certain configurations. However, using *NewBdimRouting*$_1$ is counter-productive in a few cases as using it can slightly reduce the overall throughput, by 1-2%; nevertheless, the average improvement is by over 55% and the best-case scenario yields an outstanding throughput improvement of over 185%.

We also observe how the flows routed by *NewBdimRouting*$_1$ are distributed in 2-BCN($3, 6, 3, 3$), where significant gains in hop-length can be made. The bar chart, plotted in Fig. 8, showing the normalised distribution of frequency of the number of flows in links focuses on links with at least 160 flows (the proportion of links with fewer than 160 flows is implicit). Here we see that the bottleneck flow using *NewBdimRouting*$_1$ is almost one-third smaller than that of *BdimRouting* (1,120 *vs.* 1,520).

All in all, we find that the substantial improvements in terms of path hop-length gained by using *NewBdimRouting*$_1$ over *BdimRouting* are translated into similar (or even greater) improvements in terms of network throughput.



(a) No. of hops in 1-BCN(3, 21, 3, 3). (b) No. of hops in 2-BCN(3, 21, 3, 3).

Figure 5: Bar charts of hop-lengths for *BdimRouting*, *BFS*, and $NewBdimRouting_1$.

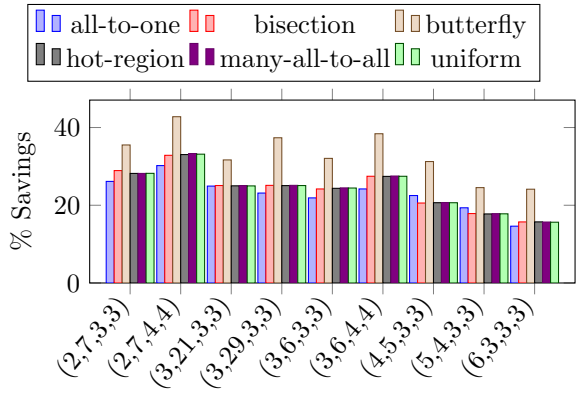


Figure 6: Mean hop-length savings with $NewBdimRouting_1$ over *BdimRouting* for different configurations of topology and traffic pattern.

6.3. Latency evaluation

We now undertake the experiments described in Section 5.5 as regards the latency incurred by using $NewBdimRouting_1$ as opposed to *BdimRouting*. We start by measuring the different phases contributing to the overall network latency.

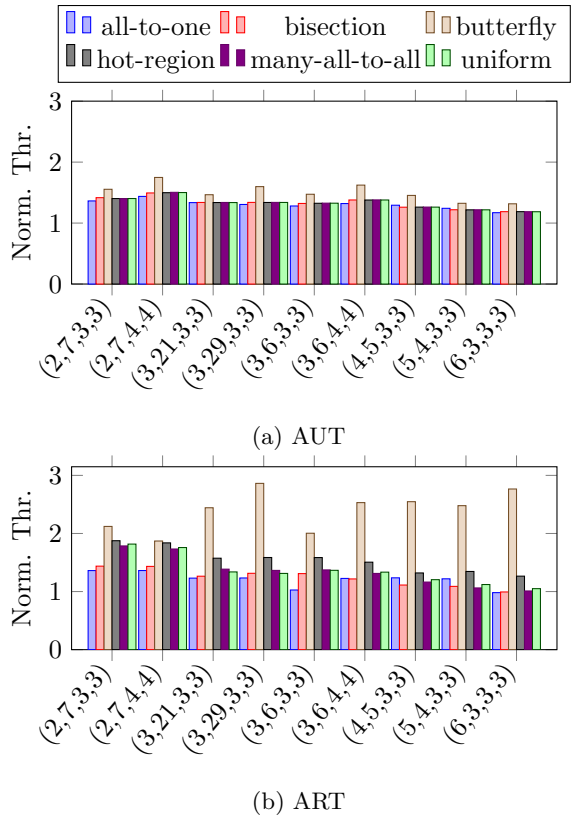
- The stack latency, L_s, is derived by measuring the round trip time of both an *empty* frame (28 bytes for the headers) and a *full* frame (1,500 bytes, including the headers) sent to `localhost`. In both cases L_s is 10μs.
- To derive the propagation latency, L_p, we measure the round trip of an empty frame sent to another server connected to the same 1Gbit Ethernet switch; this is 64μs. Dividing by two and subtracting L_s, we get an estimate of 22μs for the propagation latency.
- Similarly, we derive the data transfer latency, L_d, by measuring the round trip time of a full-frame sent to the same neighbour server; this is 140μs. Similarly, dividing by two and subtracting L_p and L_s, we get 38μs per full frame for the data transfer latency.
- We measure the average per hop running time of each algorithm, L_r, for each of the topologies when delivering a million random flows. Our measuring framework has a time resolution of nanoseconds.

Adding these measurements, we can compute the per-hop latency, $L_H = L_s + L_p + L_d + L_r$. Multiplying L_H by the average path hop-length for each algorithm gives us an estimation of the zero-load routing latency for the different routing algorithms and topologies as per Table 2 (where the data result from the generation of 1 million uniform-random flows). These experiments show how in most cases, the improvements in path length result in lower latencies; up to a 30% reduction with an average of 10%. There are, however, a couple of configurations where applying $NewBdimRouting_1$ is slightly counter-productive in terms of latency: the latency of $NewBdimRouting_1$ in BCN(6,3,3,3) and BCN(3,29,3,3) is 5% and 7% greater, respectively, than that of *BdimRouting*. The reason for this slowdown is that the number of proxies to test is much larger in $NewBdimRouting_1$ than in *BdimRouting* (over 600 in these two cases). This, in turn, renders the routing latency as dominant.

Nevertheless, note that, as explained above, using more conservative values for transmission times as well as a more optimised version of our code for the routing would still allow $NewBdimRouting_1$ to outperform *BdimRouting*.

6.4. Completion-time evaluation

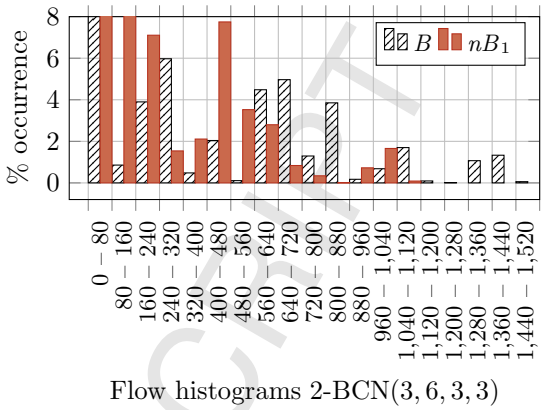
The speed up when using $NewBdimRouting_1$ for the different application models as measured with our dynamic simulation engine can be seen in Fig. 9.



(a) AUT

(b) ART

Figure 7: AUT and ART with $NewBdimRouting_1$ for different topology/traffic pattern configurations, normalized to *BdimRouting*, *i.e.*, the AUT and ART of *BdimRouting* is 1.



Flow histograms 2-BCN$(3, 6, 3, 3)$

Figure 8: Bar chart of link congestion, in number of flows, $NewBdimRouting_1$ and *BdimRouting* in the network 2-BCN$(3, 6, 3, 3)$.

BCN $(\alpha, \beta, h, \gamma)$	*Bdim-Routing*	*NewBdim-Routing*$_1$	Proxies
(2,7,3,3)	0.922 ms	0.686 ms	55
(2,7,4,4)	1.679 ms	1.164 ms	111
(3,21,3,3)	1.306 ms	1.287 ms	566
(3,29,3,3)	1.309 ms	1.410 ms	782
(3,6,3,3)	1.293 ms	1.062 ms	161
(3,6,4,4)	2.496 ms	1.984 ms	485
(4,5,3,3)	1.497 ms	1.376 ms	319
(5,4,3,3)	1.624 ms	1.625 ms	499
(6,3,3,3)	1.710 ms	1.805 ms	647

Table 2: Average latencies for *BdimRouting* and *NewBdim-Routing*$_1$ in BCN$(\alpha, \beta, h, \gamma)$, together with the number of proxies for $NewBdimRouting_1$.

These results clearly show that real applications can benefit hugely from the implementation of advanced routing schemes such as $NewBdimRouting_1$. MapReduce, an essential application in the context of DCNs, can be executed one order of magnitude faster when compared to *BdimRouting*. The other applications also obtain substantial speedups of between 1.2-3 times. In general, we can see that the lower the traffic locality and causality, and the higher its intensity, the more beneficial $NewBdimRouting_1$ becomes. MapReduce features all these characteristics and so benefits the most. Hot-region, stencil, and uniform have lower intensity and so still benefit significantly. Finally, the sweep patterns have high levels of locality and causality and thus the benefits are less noticeable.

With regards to the networks, we can see that a larger value of β makes $NewBdimRouting_1$ more beneficial because the higher diversity it offers can be better employed by its more advanced routing scheme. The only exception to this rule is hot-region; this was somehow unexpected as the throughput analysis above suggests otherwise (see Fig. 7). This exception is due to the fact that the causality introduced into the traffic does not allow the network to fully exploit its full bandwidth capabilities; so, with a non-uniform network utilization such as the one created by hot-region, the hop-length (see Fig. 6) may have a greater influence by reducing the likelihood of paths going through the more congested areas of the network. At any rate, this unexpected behaviour emphasises the need for many-dimensional studies, such as the one we perform here, that cover different aspects of the networks.

7. Conclusions

In this paper we have demonstrated, both theoretically and empirically, that there are signifi-

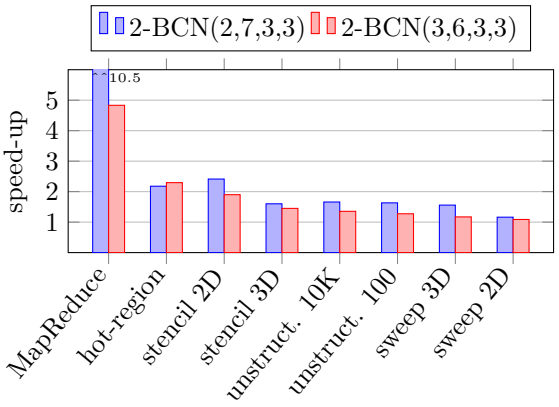


Figure 9: Completion time for eight application-like workloads in two networks. Results show the speed-up achieved using $NewBdimRouting_1$ over *BdimRouting*.

cant gains to be made as regards one-to-one routing in the DCNs HCN and BCN by using our newly-developed routing algorithms *NewFdimRouting* and *NewBdimRouting*. Moreover, in many realistic scenarios the implementation costs of employing these routing algorithms are manageable. We have benefited from an observation that the DCN HCN has (in essence) already appeared as WK-recursive interconnection networks and we have been able to utilize existing research on WK-recursive networks. Our work spawns various avenues for further research and we outline some of these now.

We have observed the general principle that shorter routes have a consequent positive effect in terms of throughput, latency, and completion time. Whilst *NewFdimRouting* is optimal in terms of the hop-lengths of the routes it finds, the algorithms encompassed within *NewBdimRouting* are not (see Figs. 5a and 5b). An obvious question is: can we improve the hop-lengths of the routes found by a one-to-one routing algorithm for BCN, so much so that these lengths are optimal? Of course, there is a tension between the complexity of a routing algorithm and the efficiency of its resulting implementation. As we have remarked, extending our current approach of exploring more proxies could well result in routing algorithms that are practically infeasible (this infeasibility might be lessened if routes that were computed had some degree of permanency associated with them and it was worth investing the effort to computer them). However, motivated by the situation as regards routing in HCN and WK-recursive networks, there could well be a combinatorial solution to this problem so that the associated combinatorics yields an efficient implementation too; that is, proxy searches can be replaced by a combinatorial analysis. This line of research provides an exciting glimpse into the hitherto mainly unexplored and exciting landscape within which modern and future datacenter networks are developed using theoretical underpinnings.

Whilst the research in this paper provides an extensive analysis of our new one-to-one routing algorithms, there is much more to routing in practical DCNs. For example, routing algorithms need to be able to tolerate faults, to balance loads, and to be energy efficient. In [15], while multiple paths between two server-nodes were shown to exist, no multi-path routing algorithm was presented. Also, the fault-tolerant routing algorithm in [15] is open to additional analysis and enhancement. As such, we need to explore whether we can develop new multi-path and fault-tolerant routing algorithms for both HCN and BCN. As a first step, there are opportunities to build upon the preliminary empirical analysis presented in [15] and to more rigorously examine the fault-tolerant routing algorithms there across a wider range of traffic patterns and workloads, as we have done in this paper.

Load balancing and energy efficiency have yet to be examined for HCN and BCN. As regards energy efficiency, this is an often overlooked aspect of DCN performance that is becoming increasingly important as the sizes of DCNs grows and more and more energy is consumed. It has been reported that datacenters accounted for 1.5% of global electricity usage in 2010 [10] with their interconnection network accounting for between 10% and 50% of this usage [1, 11]. Energy efficient routing algorithms re-route depending upon current loads on links and servers (they sometimes attempt to 'turn off' links so as to save energy) and consequently might use paths that are not always the shortest. This calls for a multi-path analysis. However, energy-efficient re-routing is only possible when there is spare capacity in the system and must be evaluated against the additional latency accrued and the energy consumed by the additional links and servers used. There is considerable scope for an examination of energy-efficient routing in HCN and BCN.

Finally, let us note the slightly surprising results we obtained as regards the performance of the two slave-connection rules we considered in this paper. We were expecting comparable performance but this was not the case. There are many more

possible slave-connection rules available for BCN (and, by extension, for DCell and FiConn) and our preliminary results here show that more research on relative performance of the various different connection rules is warranted. Not only is research needed to empirically investigate the different slave-connection rules but we need theoretical research that will tell us why one slave-connection rule should be better than another.

We close by noting that there are other aspects of routing that one might wish to evaluate that we have not considered here, such as packet loss, jitter, link quality, and so on. Some of these aspects are closer to 'real' performance but might still be evaluated through simulation. It is important that simulators are developed with the sophistication to evaluate a range of DCN properties at a reasonable scale. We intend to contribute to this in future by enhancing the functionality of our own simulation tool `INRFlow`.

Acknowledgements

This work has been funded by the Engineering and Physical Sciences Research Council (EPSRC) through grants EP/K015680/1 and EP/K015699/1. Dr. Javier Navaridas is also supported by the European Union's Horizon 2020 programme under grant agreement No. 671553 'ExaNeSt'.

References

[1] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. *SIGARCH Computer Architecture News*, 38(3):338–347, June 2010.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, Oct. 2008.

[3] P. T. Breznay and M. A. Lopez. A class of static and dynamic hierarchical interconnection networks. In *Proc. of Int. Conf. on Parallel Processing*, volume 1, pages 59–62, 1994.

[4] G.-H. Chen and D.-R. Duh. Topological properties, communication, and computation on WK-recursive networks. *Networks*, 24(6):303–317, Sep. 1994.

[5] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and A. Vasilakos. Survey on routing in data centers: insights and future directions. *IEEE Networks*, 25(4):6–10, July 2011.

[6] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan. 2008.

[7] A. Erickson, A. E. Kiasari, J. Navaridas, and I. A. Stewart. Routing algorithms for recursively-defined data centre networks. In *Proc. of Trustcom/BigDataSE/ISPA*, volume 3, pages 84–91. IEEE, Aug. 2015.

[8] A. Erickson, A. E. Kiasari, J. Pascual Saiz, J. Navaridas, and I. A. Stewart. Interconnection Networks Research Flow Evaluation Framework (INRFlow), 2016. [Software] `https://bitbucket.org/alejandroerickson/inrflow`.

[9] A. Erickson, I. A. Stewart, J. Navaridas, and A. E. Kiasari. The stellar transformation: From interconnection networks to datacenter networks. *Computer Networks*, 113:29–45, Feb. 2017.

[10] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav. It's not easy being green. In *Proc. of ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 211–222, 2012.

[11] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, Dec. 2008.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. *ACM SIGCOMM Computer Communication Review*, 39(4):51–62, Aug. 2009.

[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, Aug. 2009.

[14] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. DCell: A scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM Computer Communication Review*, 38(4):75–86, Aug. 2008.

[15] D. Guo, T. Chen, D. Li, M. Li, Y. Liu, and G. Chen. Expandable and cost-effective network structures for data centers using dual-port servers. *IEEE Transactions on Computers*, 62(7):1303–1317, July 2013.

[16] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun. Green-marl: A DSL for easy and efficient graph analysis. *SIGARCH Computer Architecture News*, 40(1):349–362, Mar. 2012.

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

[18] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, S. Lu, and J. Wu. Scalable and cost-effective interconnection of data-center servers using dual server ports. *IEEE/ACM Transactions on Networking*, 19(1):102–114, Feb. 2011.

[19] Y. Liu, J. K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi. *Data Center Networks: Topologies, Architectures and Fault-Tolerance Characteristics*. Springer, 2013.

[20] J. Navaridas, J. Miguel-Alonso, and F. Ridruejo. On synthesizing workloads emulating MPI applications. In *Proc. of IEEE Int. Symp. on Parallel and Distributed Processing*, pages 1–8, 2008.

[21] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A scalable fault-tolerant layer-2 data center network fabric. *ACM SIGCOMM Computer Communication Review*, 39(4):39–50, Oct.

2009.

[22] S. Perarnau and M. Sato. Victim selection and distributed work stealing performance: A case study. In *Proc. of 28th Int. Parallel and Distributed Processing Symp.*, pages 659–668, 2014.

[23] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network. *ACM SIGCOMM Computer Communication Review*, 45(4):183–197, Oct. 2015.

[24] I. A. Stewart. Improved routing in the data centre networks HCN and BCN. In *Proc. of Second Int. Symp. on Computing and Networking*, pages 212–218, 2014.

[25] G. D. Vecchia and C. Sanges. Recursively scalable networks for message passing architectures. In *Proc. of Int. Conf. on Parallel Processing and Applications*, pages 33–40, 1987.

[26] T. White. *Hadoop: the Definitive Guide*. O'Reilly Media, Inc., 2009.

[27] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang. MDCube: A high performance network structure for modular data center interconnection. In *Proc. of 5th Int. Conf. on Emerging Networking Experiments and Technologies*, pages 25–36, 2009.

[28] X. Yuan, S. Mahapatra, M. Lang, and S. Pakin. LFTI: A new performance metric for assessing interconnect designs for extreme-scale HPC systems. In *Proc. of 28th IEEE Int. Parallel and Distributed Processing Symp.*, pages 273–282, 2014.

[29] X. Yuan, S. Mahapatra, W. Nienaber, S. Pakin, and M. Lang. A new routing scheme for Jellyfish and its performance with HPC workloads. In *Proc. of Int. Conf. on High Performance Computing, Networking, Storage and Analysis*, pages 36:1–36:11, 2013.

[30] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proc. of 2nd USENIX Conf. on Hot Topics in Cloud Computing*, 2010.

Alejandro Erickson biography

Alejandro Erickson completed a 3-year postdoctoral research position at Durham University, United Kingdom in 2016, where he did research on various topological aspects of interconnection networks, with an emphasis on applications in datacenter networks. He received his Ph.D. in Computer Science from the University of Victoria, Canada in 2013 and his M.Math in Combinatorics and Optimization from the University of Waterloo, Canada in 2008. Dr. Erickson has

published in a broad range of topics, including datacenter networks, computational geometry, graph and matroid theory, enumerative combinatorics, education, and mathematical art.

Iain Stewart biography

Iain A. Stewart received the MA and PhD degrees in mathematics from the University of Oxford, United Kingdom in 1983 and the University of London, United Kingdom in 1986. He is a professor in the School of Engineering and Computing Sciences, Durham University, United Kingdom. His research interests include interconnection networks for parallel and distributed computing, computational complexity and finite model theory, algorithmic and structural graph theory, theoretical aspects of artificial intelligence, GPGPU computing, and computational aspects of group theory.

Jose Pascual biography

Jose A. Pascual obtained his M.Eng and Ph.D. in Computer Science at the Department of Computer Architecture and Technology of the University of the Basque Country UPV/EHU. He is currently a postdoctoral researcher at The University of Manchester. His research interests include high-performance computing, scheduling for parallel processing, and performance evaluation of parallel systems.

Javier Navaridas biography

Dr. Javier Navaridas is a Lecturer in computer architecture in the University of Manchester. Javier obtained his MEng in Computer Engineering in 2005 and his PhD in Computer Engineering (Extraordinary Doctorate Award - top 5% theses) in 2009, both from the University of the Basque Country, Spain. Afterwards he joined the University of

Manchester with a prestigious Royal Society Newton fellowship. Javier has a long publication record with more than 40 papers on interconnects, parallel and distributed systems, computer architecture, performance evaluation and characterization of application's behaviour. Javier is currently leading the workpackage on interconnects of the ExaNeSt European project.

Highlights

Improved routing algorithms for the datacenter networks HCN and BCN are proposed.
New routing algorithms are derived from algorithms for WK-recursive networks.
Routing algorithms are simulated for a variety of traffic patterns and workloads.
Our routing algorithms massively improve on existing ones.