Hyper-BCube: A Scalable Data Center Network

Dong Lin, Yang Liu, Mounir Hamdi and Jogesh Muppala

Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong
{ldcse, liuyangcse, hamdi, muppala}@cse.ust.hk

***Abstract*—Mega data centers are being built around the world to provide various cloud computing services. As a result, data center networking has recently been a hot research topic in both academia and industry. A fundamental challenge in this research is the design of the data center network that interconnects the massive number of servers, and provides efficient and fault-tolerant routing service to upper-layer applications. In response to this challenge, the research community have begun exploring novel interconnect topologies including Fat-Tree, DCell, and BCube and etc. Understandably, this research is still in its infancy. The proposed solutions either scale too fast (i.e., double exponentially) or too slow, suffer from performance bottlenecks, or can be quite costly in both routing and construction. In this paper, we propose a cost-effective and gracefully scalable data center interconnect termed Hyper-BCube that combines the advantages of both DCell and BCube architectures while avoiding their limitations. We then propose fault-tolerant and routing mechanisms for Hyper-BCube. Finally, we propose a comprehensive benchmarking environment that can be used for accurately and practically evaluating and testing the proposed data center architecture.**

Keywords-Data center, Network topology, Throughput, Fault tolerance

I. INTRODUCTION

Data center infrastructure design has recently been receiving significant research interest both from academia and industry, in no small part due to the growing importance of data centers in supporting and sustaining the rapidly growing Internet-based applications including search (e.g. Google), online video (e.g. YouTube), social networking (e.g. Facebook), and large-scale computations (e.g. data mining, bioinformatics).

In particular, Cloud computing (e.g. Amazon EC2, Google App Engine, and Microsoft Azure) is touted as the culmination of the integration of computing and data infrastructures to provide a scalable, agile and cost-effective approach to support the ever-growing IT needs of both enterprises and the general public [2][11]. Massive data centers providing the core support infrastructure for the cloud, amounting up to 45% of the total cost [2]. It is thus imperative that we get the data center infrastructure, including the data center networking right so that both the deployment and maintenance of the infrastructure is cost-effective.

With data availability and security at stake, the role of the data center is more critical than ever. This is seen in the increase of data center expenditures, new standards being established, and the increased role of the data center. According to industry estimates, the US data center market reached almost US$39 billion in 2009, growing from US$16.2 billion in 2005. This market trend is seen all over the world. For example, Taiwan is planning to create a complete supply chain incorporating a cloud computing-based Internet data center. The entire supply chain will be sold to overseas markets and is expected to gain sales and business opportunities of US$10 billion by 2014 [7].

Traditionally data center networking was based around top of rack (ToR) switches interconnected through end of rack (EoR) switches, and these in turn being connected through core switches. This approach, besides being very costly, leads to significant bandwidth oversubscription towards the network core. This prompted several researchers to suggest alternate approaches for scalable cost-effective network infrastructures, based on topologies including Fat-Tree [2][11][15], DCell[3], BCube[4], MDCube[8], and Clos networks[1].

While these initial research efforts are a step in the right direction, there are still many outstanding and challenging issues that need to be addressed. For example, the proposed solutions either scale too fast (i.e. double exponentially) or too slow, have performance bottlenecks, or can be quite costly.

The motivation in this paper is to provide solutions to these challenges where the ultimate goal is to build a scalable cost-effective data center networking infrastructure. In particular, we propose a data center interconnect, termed Hyper-BCube, which scales in a flexible way using small commodity switches. It combines the advantages of both DCell and BCube architectures while avoiding their drawbacks.

The rest of the paper is organized as follows. In Section II, we briefly review the related work from the literature. In Section III, we describe the structure of Hyper-BCube. We then propose routing, load-balancing, and fault-tolerant mechanisms for this new architecture. In Section IV, we provide a comprehensive benchmarking environment to evaluate the proposed data center infrastructure. Finally, we conclude our work in Section V.

II. RELATED WORK

Numerous proposals for identifying suitable network architectures for massive data centers have been investigated and implemented in both academia and industry. These *Data Center Interconnects* (DCIs) can be classified based on whether they evolved from the field of parallel computing or whether they evolved based on Internet switches and routers.

This research has been supported by a grant from Huawei Technologies Co. LTD. (Project code: HUAW10B15Z002.09/10PN).

A. DCIs Evolving from Parallel Computing

Several recently proposed scalable and fault-tolerant data center networking architectures such as DCell [3], FiConn [6], BCube [4] and MDCube [8] build upon the rich research literature on the interconnection networks derived for parallel computations [10] including some earlier research [12][13][14].

DCell is a recursively defined architecture. Servers in DCell have multiple ports. Each server is connected with a single mini-switch and with many other servers via communication links. $DCell_0$ is the basic building block to construct a larger DCell. It consists of n servers and they are connected to a n-port switch. $DCell_k$ is formed using a_{k-1}+1 $DCell_{k-1}$s, where a_{k-1} denotes the number of servers in a $DCell_{k-1}$. As a result, the DCell architecture scales double exponentially.

FiConn is another example of a recursive structure. It requires at most 2 ports on computers, while still can scale up to millions of computers. The basic element of FiConn, the $FiConn_0$, is the same as $DCell_0$. Each computer in FiConn has a port connected to the switch in its $FiConn_0$, which is called level-0 port. The backup port on the computer is left to connect to other computers. Assume that there are b available backup ports inside a $FiConn_{k-1}$, to construct a $FiConn_k$, one needs to connect $b/2$ available ports to $b/2$ other $FiConn_{k-1}$s. As a result, a $FiConn_k$ consists of (b/2+1) $FiConn_{k-1}$s.

BCube is a server-centric network structure. There are two types of devices which forms the BCube structure - servers with multiple network ports and mini-switches which connect servers at different layers. A $BCube_k$ is constructed recursively from n $BCube_{k-1}$s and n^{k-1} n-port switches. In a BCube structure, switches never directly connect to other switches and they just do forwarding. The BCube uses a lot of wires and switches and has high cabling complexity that prohibits it from being scaled beyond a shipping container based modular data center (MDC).

The MDCube [8] is an attempt at scaling up a BCube-based MDC to a large number of servers, using the MDC as a building block. In such a structure, each container is labeled with an m-tuple. A container is connected to all the containers that have only one different digit in their labels.

The PortLand architecture is designed on a multi-rooted network topology which is referred to Fat-Tree [15]. It is known to have performance bottlenecks, and has poor fault-tolerance topological properties. For example, if the centralized fabric manager fails the whole PortLand scheme will fail.

B. DCIs Evolving from Internet Switches and Routers

Many proposals for data center networking architectures are based on architectures originally designed for Internet switches and routers. These include VL2 [1] and DOS [16].

A VL2 network is built from multiple switches arranged into a Clos topology to support large path diversity. Using valiant load balancing to spread destination independent traffic among multiple servers, VL2 provides uniform high capacity between any two servers. However, its scalability, that is the number of servers, is restricted by the maximum port count of an intermediate switch.

DOS uses wavelength routing characteristics based on an array waveguide grating router to design a scalable optical switch for data centers. They take advantage of having multiple wavelengths to demonstrate that their architecture outperforms electronic switches.

III. HYPER-BCUBE

A. Motivation

In general, data center network architectures can be benchmarked by using scalability, availability and complexity together to achieve a comprehensive evaluation. In practice, these three requirements usually appear to be in conflict with each other. The design of data center network architecture itself contains a series of tradeoffs. In particular, scalability has significant influence on the performance and the cost of entire data center network, and therefore is highlighted. Despite of some differences, previous architectures/algorithms can be classified into three categories according to their scalabilities (i.e. the number of servers under certain configuration) : $O(n^c)$ [1][2][11][15], $O(n^k)$[4] and $O(n^{(2^k)})$ [3][6], where c refers to a constant; n is the port-count of switches/routers and k denotes the node/server degree or the number of network layers.

Scalable data center network architecture must be capable of hosting millions of servers by using minimal costs in terms of interface cards, intermediate switches and etc. Moreover, it must yield gradual performance in a fault-tolerant environment. Many traditional data center network architectures although provide excellent scalability (e.g. $O(n^{(2^k)})$ [3][6]), fail to yield a gradually scalable performance. In practice, it is common to leave part of the data center network empty to ease future extension. Such partially equipped data center is usually highly inefficient, especially for those double-exponentially-scaled data center networks. E.g. one step further from 1K-server data center turns to be 1M-server scale.

On the other hand, $O(n^k)$ and $O(n^c)$ – scaled speed are inadequate in practice as well, given the following practical observations:

- The maximum-allowed node degree (i.e. k) is typically less than 6. This matches the physical restriction for servers in practice having less than six interface cards.
- The port count of a switch (i.e. n) is strictly restricted and small value (e.g., 4 to 8) is preferred. Because the price of a switch typically increases much faster than its number of ports [9]. A small-port-count switch is always more cost-effective than larger size switches.

With the above observations in mind, an exponentially scaled data center network like BCube can have a scalability problem. For example, with n = 4, it requires 6 layers to construct a data center with 4^6=4096 servers which falls far behind the need of today's data center network.

In this paper, we intend to investigate the scalability problem from a different angle – *what kind of scalability is favorable given small-port-count of intermediate switches and restricted node/server degree?* How about $O((an)^k)$? Here "a" is a constant comparable to "n".

B. Hyper-BCube

As the first attempt to explore the advantage of an $O((an)^k)$-class DCI, we start from one special case where $a = n$. In this paper, we propose a new interconnection network architecture called Hyper-BCube which scales at a speed of $O(n^{2k})$.

The first layer of the Hyper-BCube contains n nodes and one n-port switch. It is the same as that of a $DCell_0$. Starting from the second layer, a k-layer (k≥2) Hyper-BCube consists of n^2 (k-1)-layer Hyper-BCubes. In particular, a k-layer (k≥2) Hyper-BCube can simply be regarded as an $n^2 * n^{2k-3}$ matrix, where each row is a (k-1)-layer Hyper-BCube which consists of n^{2k-3} nodes. Alternatively, it can be considered as having n^{2k-3} columns and each column contains exactly n^2 nodes which belong to n^2 (k-1)-layer Hyper-BCubes respectively. Column-based connection is introduced here to connect the n^2 nodes located at the same column by using exactly n n-port switches. The connection patterns of these n switches are listed as shown in Table I. There are only two different kinds of connection patterns. And they always interleave with each other. Fig. 1 demonstrates the connection pattern of the first two columns in a k-layer (k≥2) Hyper-BCube with 4-port switches.

C. Routing and Fault-tolerance

The local re-routing algorithm adopted by DCell and others has been proved to be inefficient [5]. A fault-free routing path could transit through a specific layer of network repeatedly leading to a path length of $O(2^k)$. In contrast, routing in Hyper-BCube is more efficient as a fault-free routing path typically transits through a k-layer (k>1) Hyper-Cube for only once.

A node in a k-layer Hyper-BCube can be labeled using k coordinates, (C_k, C_{k-1}, ..., C_1), where "C_j" denotes that this node is located at the C_j-th row of a j-layer Hyper-BCube. We use a hierarchical row-based routing for a Hyper-BCube. So given a pair of nodes, a path between the source (S_k, S_{k-1}, ..., S_1) and the destination (D_k, D_{k-1}, ..., D_1) can be established through the following k steps, where only one coordinate is used in each step: (S_k, S_{k-1}, ..., S_1) →(D_k, ?, ..., ?) → (D_k, D_{k-1}, ..., ?) →…→ (D_k, D_{k-1}, ..., D_2,?) → (D_k, D_{k-1}, ..., D_2, D_1). The "?" denotes unknown/don't care value.

Each step may further require multiple transitions. Taking one single step routing that from (...., S_j, ...) to (...., D_j, ...) as an example, we demonstrate how to find a fault-free routing path from the S_j-th row to the D_j-th row. First, S_j and D_j can be represented as follows.

$$F(z) = \begin{cases} n - (z \bmod n) & if\ (z \bmod n \neq 0) \\ 0 & if\ (z \bmod n = 0) \end{cases}$$

$$S_j = \left\lfloor \frac{S_j}{n} \right\rfloor * n - F(S_j)$$

$$D_j = \left\lfloor \frac{D_j}{n} \right\rfloor * n - F(D_j)$$

Thus, the distance between the S_j-th row and the D_j-th row can be represented as following.

$$D_j - S_j = \left(\left\lfloor \frac{D_j}{n} \right\rfloor - \left\lfloor \frac{S_j}{n} \right\rfloor\right) * n - \left[F(D_j) - F(S_j)\right]$$

Without link failure, according to the values of $\left[F(D_j) - F(S_j)\right]$ and$(\lceil D_j/n \rceil - \lceil S_j/n \rceil)$, a fault-free path from the S_j-th row to the D_j-th row can be established through zero to three

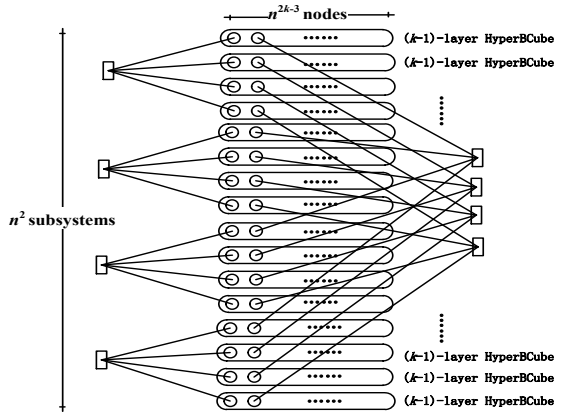

Figure 1. The connection pattern of the first two columns of a k-layer (k≥2) Hyper-BCube with 4-port switches. Please note that *only* the k-th layer's switches are illustrated here.

TABLE I. THE DESTINATION ROW NUMBER OF THE CORRESPONDING SWITCH FOR THE I -TH COLUMN

	j-th port, (*i* mod 2)=1	*j-th port*, (*i* mod 2)=0
***x*-th switch**	**(*x*-1)*n*+*j***	**(*j*-1)*n*+*x***

* **(1≤*x*≤*n*); (1≤*j*≤*n*); (1≤*i*≤n^{2k-1}) *k*≥1**

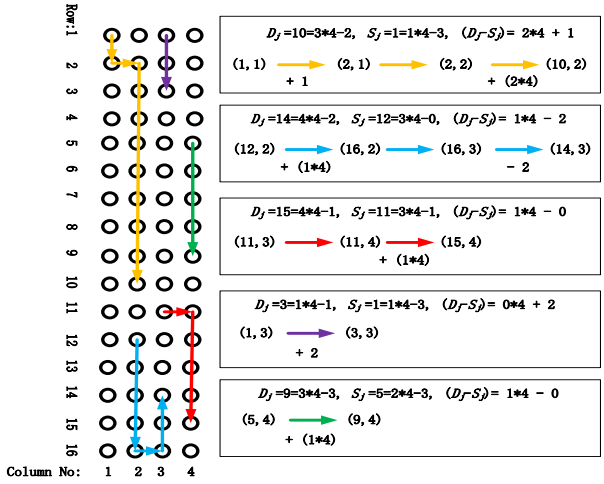

Figure 2. Fault-Free routing in a 2-layer Hyper-BCube with 4-port switches. There are 64 nodes in total.

Single-Step-Routing(*S*$_j$, *D*$_j$, *j*, *RetryTimes*){
if (*RetryTimes*==0) return null;
if (*S*$_j$==*D*$_j$) return a 0-hop path;
else{
Try 1-hop paths in sequence through a directly connected *j*-layer switch, if succeed, return a 1-hop path;
Try 2-hop paths in sequence, first through a directly connected 1-layer switch, further through a *j*-layer switch, if succeed, return a 2-hop path;
Using 1-hop and 2-hop reachable nodes as intermediates, try Single-Step-Routing (intermediate, *D*$_j$, *j*, *RetryTimes*-1), if succeed, return a combined path;
if all fail, return null;
}
}

Routing(*Src*, *Dest*, *k*, *RetryTimes*){
Call Single-Step-Routing (*S*$_j$, *D*$_j$, *j*, *RetryTimes*) for *k* times;
For each single step routing, gradually increase the value of *RetryTimes*, so as to guarantee shortest length of feasible paths.
}

Figure 3. The routing algorithm of Hyper-BCube.

transition(s) in two adjacent columns. For example, when($\lceil D_j/n \rceil - \lceil S_j/n \rceil) \neq 0$, one transition through the i-th column is required, where (i mod 2) = 0. Similarly, when $\lceil F(D_j) - F(S_j) \rceil \neq 0$, one transition through the i-th column is required, where (i mod 2) = 1. Moreover, when both of the above two transitions are required, one more transition through the i-th column to the $(i \pm 1)$ -th column is required. Therefore, at most three transitions at the j-th (j>1) layer of a Hyper-BCube are required in order to secure one coordinate. We present several examples in Fig. 2 where paths between five pairs of nodes are established. As shown in Fig.2, the required number of transitions varies from one to three. Given the time complexity of one step routing is $O(1)$, the total time complexity of fault-free routing in k-layer Hyper-BCube is $O(k)$.

In Hyper-BCube architecture, fault-tolerant routing can be implemented by using alternative paths that can be found by conducting routing through various layers of the network. In Hyper-BCube, there are only two different column-based connection patterns which interleave with each other. Accordingly, in a k-layer (k≥2) Hyper-BCube network with n^2*n^{2k-3} nodes, there are least ($n^{2k-3}/2$) identical and parallel paths between any pair of rows. In other words, there must be an alternative path located at $2m$ columns away from current column. The value of m determines which layer of network has been selected to conducting such bypass.

Fig.3 gives the fault-tolerant routing algorithm of Hyper-BCube, which consists of k steps single-step-routing. Each single-step-routing is recursively defined. It returns a feasible path with maximal $2^{RetryTimes}$ hops. Given multiple feasible paths, it chooses the first available path in sequence. For instance, it always starts from the next port of current switch. For each single-step-routing, the value of *RetryTimes* is increased gradually to guarantee the shortest length of feasible paths. The entire routing algorithm of Hyper-BCube can be regards as a restricted breadth first search (BFS). Unlike conventional BFS adopted by BCube which entails $O(|V|+|E|) = O((k'+1)|V|)$ complexity, the complexity of the fault-tolerant routing algorithm of Hyper-BCube is at most $O(k*n^{2RetryTimes}) = O(|V|*kn^{2(RetryTimes-k)+1})(k'=2k-1)$, due to its non-rollback routing scheme. (i.e., the previous successfully routed single-step-routing will not be started over again so as to find an alternative path.) Given small values of n and *RetryTimes* (e.g. $<<k$), such complexity in practice is trivial and outperforms previous algorithms, such as DCell and BCube.

D. Key Features of Hyper-BCube

Our preliminary investigation reveals that the Hyper-BCube topology exhibits some good properties that strike a good compromise between the excessive scalability of DCell and high-cost of BCube as it is show in Table II.

a) Diameter: The shortest distance between any nodes.

The diameter of a Hyper-BCube network is at most ($3k$-2). Because the number of transitions for each step (i.e. the procedure of securing one coordinate) is at most three. Thus, the total number of transitions above the k-th layer of a Hyper-BCube is at most $3(k-1)$. With additional one transitions in a 1-layer Hyper-BCube, the diameter of a Hyper-BCube is at most ($3k$-2). Given an appropriate value of k, Hyper-BCube is superior to DCell in terms of routing path length and diameter.

TABLE II. PROPERTIES OF DIFFERENT NETWORK ARCHITECTURES

	DCell	***BCube***	***Hyper-BCube***
Number of nodes	$a_1 = n$ (k=1) $a_k = a_{k-1}*(a_{k-1}+1)$ (k≥2)	n^k	n^{2k-1}
Node degree	k	k	k
Number of Links	$(k+1)*a_k/2$	kn^k	kn^{2k-1}
Number of Switches	a_k/n	kn^{k-1}	kn^{2k-2}
Diameter	2^k-1	k	≤($3k$-2)
Bisection Bandwidth	$a_k/(4log_n a_k)$	$n^k/2$	$n^{2k-1}/4$ (k≥2)

The number of ports of a switch is n, and there are k layers networks.

b) Bisection Bandwidth: the minimum number of links cut when a network is partitioned into two equal halves over all partitions.

The bisection bandwidth of a k-layer Hyper-BCube network is ($n^{2k-1}/4$). It is attractive, as the bisection bandwidth of the Hyper-BCube network grows exceptionally and matches the increasing speed of node number. In contrast, the bisection bandwidth of DCell increases sub-linearly to the increasing speed of node number.

c) Incremental construction: a Hyper-BCube DCI can be built partially so as to support incremental construction. The architecture of Hyper-BCube is highly symmetric. Missing entire rows/columns will not reduce the system performance significantly. For example, given absence of 1/n rows (e.g. those mod n=1), the entire network can be still regards as a complete Hyper-BCube which is constructed by using (n-1)-port switches. Similarly, the repetitive column-based connections always provide us with multiple alternative paths.

d) Speedup of the 1st layer: Hyper-BCube tries to allocate most of the traffic to its bottom layer. The routing algorithm of Hyper-BCube requires multiple transitions through 1st layer network. In the most extreme cases, the traffic intensity of the 1st layer network could be k times that of in the other layers.

In practice, the speedup factor of the 1st layer network is typically less than 3 for a data center with millions of servers as will be demonstrated latter. Moreover, since the size of 1st layer network is extremely small (e.g. n nodes) and these nodes are always directly connected; such speedup requirement can be easily satisfied by using dedicated high-speed short-distance interface cards (e.g. InfiniBand) or virtual machine technology.

IV. SIMULATION

In this section, we use simulations to evaluate the performance of Hyper-BCube and make it compare with that of DCell and BCube.

A. Average Path Length under Different Configurations

First, we study their practical average path length (APL) between any random pair of nodes/servers under different configurations. Table III summarizes the APLs of Hyper-BCube, DCell and BCube under various configurations. Both BCube and Hyper-BCube outperform DCell by yielding a much shorter average path length (APL). To build a data center with 176820 nodes, APL of DCell is 11.29. In contrast, the APL of a larger data center with 279936 nodes using BCube/ Hyper-BCube is only 5.83/8.33. Besides, the APLs for both BCube and Hyper-BCube are increased linearly. For example, the APL of Hyper-BCube increases linearly to the node degree

TABLE III. THE CHARICTISTICS UNDER DIFFERENT NETWORK CONFIGURATIONS (FAULT-FREE)

	DCell					*BCube*					*Hyper-BCube*				
	Nodes	*Links*	*APL*	*ABT*1	ABT2	*Nodes*	*Links*	*APL*	*ABT*1	ABT2	*Nodes*	*Links*	*APL*	*ABT*1	*ABT*2
***n*=4, *k*=2**	**20**	**30**	**2.26**	**14.6**	**4**	**16**	**32**	**1.6**	**20**	**4**	**64**	**128**	**3.05**	**60**	**26.8**
***n*=4, *k*=3**	**420**	**840**	**5.16**	**161**	**65.7**	**64**	**192**	**2.29**	**84**	**20**	**1024**	**3072**	**5.26**	**693**	**234**
***n*=4, *k*=4**	**176820**	**442050**	**11.29**	**33589**	**16353.5**	**256**	**1024**	**3.01**	**340**	**66**	**16384**	**65536**	**7.5**	**10930**	**3514**
***n*=4, *k*=5**	**≈3.1e10**	**≈9.4e10**				**1024**	**5120**	**3.75**	**1364**	**277**	**262144**	**1310720**	**9.75**	**174768**	**55832**
***n*=4, *k*=6**						**4096**	**24576**	**4.5**	**5460**	**1153**	**4194304**	**25165824**	**12**		
***n*=4, *k*=7**						**16384**	**114688**	**5.25**	**21844**	**4515**					
***n*=4, *k*=8**						**65536**	**524288**	**6**	**87380**	**18068**					
***n*=4, *k*=9**						**262144**	**2359296**	**6.75**	**349520**	**72046**					
***n*=6, *k*=2**	**42**	**63**	**2.46**	**27**	**4**	**36**	**72**	**1.71**	**42**	**4**	**216**	**432**	**3.35**	**156**	**35.9**
***n*=6, *k*=3**	**1806**	**3612**	**5.73**	**592**	**138.6**	**216**	**648**	**2.51**	**258**	**30**	**7776**	**23328**	**5.83**	**4682**	**827**
***n*=6, *k*=4**	**≈3.3e6**	**≈8.2e6**				**1296**	**5184**	**3.34**	**1554**	**155**	**279936**	**1119744**	**8.33**	**167972**	**29411**
***n*=6, *k*=5**						**7776**	**38880**	**4.17**	**9330**	**975**	**10077696**	**50388480**	**10.83**		
***n*=6, *k*=6**						**46656**	**279936**	**5**	**55980**	**5798**					
***n*=6, *k*=7**						**279936**	**1959552**	**5.83**	**335880**	**34514**					

^ All the links are capable of two-way communication. The bandwidth of a link in one-way communication is "1" by default. Identical links are used for all DCIs, except that the bandwidth of a 1st layer link in a Hyper-BCube is $\lceil k/2 \rceil$.

The traffic pattern for ABT1 and ABT2 are all-to-all and some-to-all respectively. In all-to-all communication, all nodes are active. Each node communicates with all nodes (except itself) in two-way communication [4]. In some-to-all communication, only $1/x$ nodes are selected randomly. They communicate with all nodes (except itself) in two-way communication. By default, $x=n^2$ for BCube and Hyper-BCube; $x=\varepsilon$ for DCell, where ε refers to the number of servers in a $DCell_1$. E.g. $DCell_1$=20 for n=4.

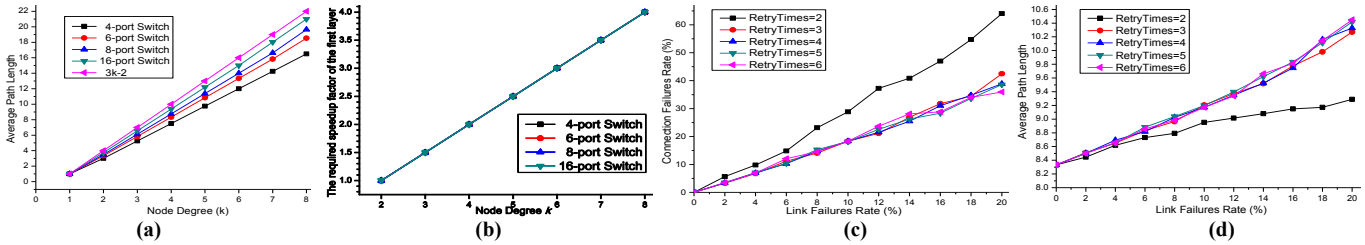

Figure 5. Key features of Hyper-BCube under different configurations.

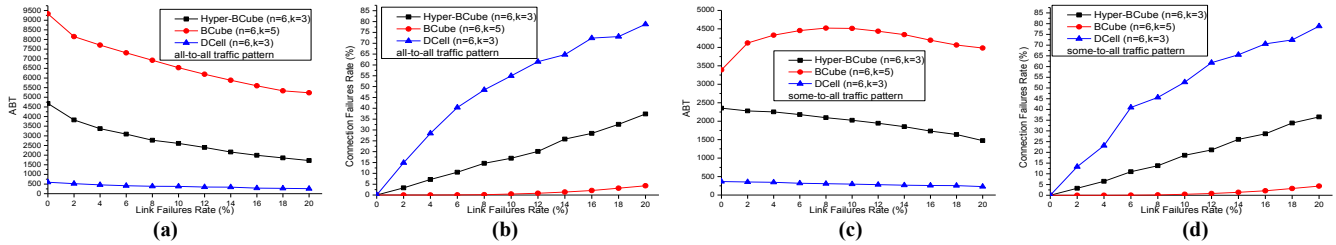

Figure 6. Performance comparisons under faulty environment.

k as show in Fig. 5(a), where the port-count of switches only affects the gradient slightly. Thus, BCube and Hyper-BCube actually entail better scalability in terms of average path length.

B. *ABT under Fault-free*

Hyper-BCube is a symmetric structure that each node has exactly one link to the j-th layer Hyper-BCube, making the total link number of different layers the same. In this way, Hyper-BCube avoids the bottleneck effect.

Due to the routing scheme that has been adopted, Hyper-BCube usually has a heavily loaded 1st layer. Fig. 5(b) presents the required speedup factor of the 1st layer of a Hyper-BCube under all-to-all traffic pattern. The required speedup does not affect by the port-count of intermediate switches but increases linearly with the node degree at a speed of $k/2$. For a million-server data center, such value is typically less than 3. For example, with 6-port switches and k=5, the number of servers is 10 million, the required speedup factor of the 1st layer Hyper-BCube is only 2.5.

Given *just-enough* speedup (i.e.$\lceil k/2 \rceil$) to the 1st layer of a Hyper-BCube, we test the Aggregation-Bottleneck-Throughput (ABT) [4] of three DCIs under various configurations. Both all-to-all and some-to-all communication patterns are generated here for the sake of comprehensive evaluations.

ABT measures the maximal sustainable throughput over the entire DCI. Given multiple flows over a link, the bandwidth of this link is shared by these flows uniformly. And the maximal throughput between any pair of nodes is restricted to the bottleneck throughput of the corresponding routing path. As shown in Table III, both BCube and Hyper-BCube provide much higher ABT than DCell. For example, the ABT1 of a BCube / Hyper-BCube with 262144 servers is 349520 / 174767, which is roughly 10 / 5 times that of a 176820-server DCell (i.e. 33589). On the other hand, with smaller node degree, Hyper-BCube still provides equivalent scalability and ABT per link performance, leading to a more cost-effective architecture for large-scale data center. In particular, the ABTs of Hyper-BCube and BCube are very close to each other under some-to-all traffic patterns. For example, for a data center with 279936 servers, the ABT2 of Hyper-BCube is (29411/34514≈85.2%) that of BCube, while requiring (1119744/1959552≈57.1%) of the connection cost. In practice, the some-to-all traffic pattern matches the mapping and reducing phases of a map-reduce application, and therefore is usually highlighted.

C. Fault-tolerant Routing Complexity

One merit of Hyper-BCube lies in its low time complexity of fault-tolerant routing (i.e. $O(|V|*kn^{2(RetryTimes-k)+1})$). In order to understand the influence of *RetryTimes*, we choose a fixed configuration of Hyper-BCube where n=6, k=4 and with all-to-all traffic pattern. We increase the link failure rate gradually to check the impact on the connectivity and APL. Fig. 5 (c) and (d) illustrate the corresponding results. Although the results do not show perfect monotonicity due to the sampling scheme, the tendency is quite clear that the Hyper-BCube suffers from linearly decayed performance under faulty condition. And a larger value of *RetryTimes* only improves the performance slightly. Set *RetryTimes* to 3 could achieve a perfect balance between the capability of fault-tolerance and the complexity of routing algorithm. Accordingly, the time complexity of fault-tolerant routing in a Hyper-BCube is only 1/[2*n^(2k-7)] that of in a BCube. For k>3, Hyper-BCube outperforms BCube in terms of routing complexity. E.g. the routing complexity of a Hyper-BCube (n=4, k=5) is only 1/128 that of a BCube (n=4, k=9), while both of them hosting 262144 servers.

D. ABT and connectivity under Faulty

ABT and the connectivity of DCIs under faulty condition are extremely important. We examine these two features of different DCIs in this part.

Given the exponentially increasing complexity of BCube's fault-tolerant routing, we can only study its ABT under faulty condition in small scale. E.g. a BCube (n=6, k=5) with 7776 servers reaches the limit of computation complexity.

In this specific simulation, DCell (6, 3), BCube (6, 5) and Hyper-BCube (6, 3) are chosen to compare against each other, which host 1806, 7776 and 7776 servers respectively. It is not a fair comparison in terms of the number of servers; however the closest one that is available to us.

As illustrated in Fig. 6, for both all-to-all and some-to-all (i.e. 1/n - to - all) traffic partners, BCube outperforms its competitors by providing a higher ABT and lower connection failure rate. In particular, we observe in Fig. 6 (c) that the ABT of the BCube under some-to-all traffic pattern actually rises up a little before dropping down. The explanation of this phenomenon lies in the BFS algorithm that BCube has adopted for its fault-tolerant routing. The BFS algorithm always chooses the shortest feasible path which differs from that of BCube under fault-free condition. This phenomenon also reveals that excellent performance of BCube under faulty condition actually represents the optimal scenario of "exhaustive searching in a graphic" which is impractical in large scale data center, due to the high time complexity.

In contrast, with scalable and low complexity fault-tolerant routing algorithm, the Hyper-BCube still yields a comprised performance between the BCube and the DCell.

V. CONCLUSION

In this paper, we have presented the design, implementation, and evaluation of novel network architecture named Hyper-BCube, to scale data center to mega level with only small-port-count switches and small node degree. Hyper-BCube strikes a compromise between the excessive scalability of DCell and high cost of BCube. Given an equal sized data center, the cost of Hyper-BCube in terms of number of links and switches is roughly 1/2 that of BCube, while still providing comparable overall performances. Hyper-BCube is also fault-tolerant and load-balancing in nature due to its special structure design and the low-time-complexity routing protocol on top of its network topology.

REFERENCES

[1] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Pat., "VL2: A Scalable and Flexible Data Center Network," In *SIGCOMM* 2009 on Data communication, Pages: 51-62, August 16-21, 2009, Barcelona, Spain.

[2] A. Greenberg, J. R. Hamilton, D. A. Maltz, P. Patel. "The cost of a cloud: research problems in data center networks", *ACM SIGCOMM Computer Communication Review*, Vol. 39, No. 1, Jan. 2009.

[3] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, S. Luz, "DCell: A Scalable and Fault Tolerant Network Structure for Data Centers," In *SIGCOMM* 2008 on Data communication, Pages: 75-86, August 17-22, 2008, Seattle, WA, USA.

[4] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," In *SIGCOMM* 2009 on Data communication, Pages: 63-74, August 16-21, 2009, Barcelona, Spain.

[5] Chayan Sarkar, "Seminar Report on Data Center Network Design", Technical Report, 2010, http://www.cse.iitb.ac.in/~chayan/seminar/final_report_dcn.pdf.

[6] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu "FiConn: Using Backup Port for Server Interconnection in Data Centers", IEEE INFOCOM 2009, April 19-25, 2009, Rio de Janeiro, Brazil.

[7] Green (low carbon) Data Center Blog, 2010, http://www.greenm3.com/2010/02/taiwans-cloud-computing-data-center-31-mil-usd-investment.html

[8] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: A High Performance Network Structure for Modular Data Center Interconnection", in *ACM SIGCOMM CoNEXT*, December 2009.

[9] IT247,http://www.it247.com/category/Network-Storage-Comms-Routers-Switches-Security-Routers-Switches.html

[10] L. Bhuyan and D. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," *IEEE trans. Computers*, April 1984.

[11] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable Commodity Data Center Network Architecture," In *SIGCOMM* 2008 on Data communication, Pages: 63-74, August 17-22, 2008, Seattle, WA, USA.

[12] M. Hamdi and R. W. Hall, "RCC-FULL: An Effective Network For Parallel Computations," *Journal of Parallel and Distributed Computing*, Vol. 41, no. 2, March 1997, pp. 139-155.

[13] M. Hamdi, "Recursive Interconnection Networks: Architectural Characteristics and Hardware Cost," *IEEE Transactions on Circuits and Systems*, Vol. 41, No. 12, pp. 805-816, Dec. 1994.

[14] M. Hamdi and R. W. Hall, "Image Processing on Augmented Mesh-Connected Parallel Computers," *Journal of Computer and Software Engineering*, Vol. 2, No. 3, pp. 329-348, 1994.

[15] R. N. Mysore, A. Pamporis, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable, Fault-Tolerant Layer 2 Data Center Network Fabric," In *SIGCOMM* 2009 on Data communication, Pages: 39-50, August 16-21, 2009, Barcelona, Spain.

[16] X. Ye, Y. Yin, S.J.B.Yoo, P. Mejia, R.Proietti and V. Akella, "DOS – A Scalable Optical Swtich for Datacenters", ACM *ANCS*'10, October 25-26, 2010, La Jolla, CA, USA.