Computer Communications 129 (2018) 89–100

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A cost-effective low-latency overlaid torus-based data center network architecture

Ting Wang[1,b], Lu Wang[*,2,a], Mounir Hamdi[3,c]

[a] College of Computer Science and Software Engineering, Shenzhen University, China
[b] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China
[c] College of Science, Engineering and Technology, Hamad Bin Khalifa University, Qatar

ARTICLE INFO

Keywords:
Data center network
Architecture
Torus
Network topology
Probabilistic weighted routing
Deadlock-free

ABSTRACT

In this paper, we present the design, analysis, and implementation of a novel data center network architecture named *CLOT*, which delivers significant reduction in the network diameter, network latency, and infrastructure cost. *CLOT* is built based on a switchless torus topology by adding only a number of most beneficial low-end switches in a proper way. Forming the servers in close proximity of each other in torus topology well implements the network locality. The extra layer of switches largely shortens the average routing path length of torus network, which increases the communication efficiency. We show that *CLOT* can achieve lower latency, smaller routing path length, higher bisection bandwidth and throughput, and better fault tolerance compared to both conventional hierarchical data center networks as well as the recently proposed CamCube network. Coupled with the coordinate based geographical addresses and credit based flow control, the specially designed POW routing algorithm helps *CLOT* achieve its maximum theoretical performance. Besides, an automatic address configuration mechanism and malfunction detection mechanism are provided to facilitate the network deployment and configuration. The sufficient mathematical analysis and theoretical derivation prove both guaranteed and ideal performance of *CLOT*.

1. Introduction

Serving as the core infrastructure for the cloud providers as well as large-scale enterprise applications, the data center network (DCN) plays a key role in determining the performance of service provided to users. The traditional hierarchical switched DCN topology, besides being very costly, suffers high oversubscription ratio towards higher layers leading to serious communication bottleneck [1]. Thus, researchers proposed several more cost-effective DCN architectures such as BCube [2], DCell [3], SprintNet [4,5], FlatNet [6], CamCube [7], Small-World [8] and NovaCube [9], where these server-centric architectures abandon expensive high-end network switches by using only low-end switches or even no switches at all. In addition, the forwarding functionality is shifted to the servers, which helps achieve higher bisection bandwidth and better fault tolerance with richer connections [10]. However, they suffer high latencies due to their relatively long routing path length, e.g. in torus-based architectures, and relatively poor path diversity. As a result, some optical DCN topologies like OSA [11] have been proposed. Compared with packet switching, the optical circuit switching can provide higher bandwidth and lower latency in transmission. However, the optics suffer from slow switching speed which can take as long as tens of milliseconds, and cannot achieve full bisection bandwidth at packet granularity.

Based on these observations, in this paper we propose a novel DCN architecture named *CLOT*, which is built on a k-ary n-D torus topology whose various unique advantages have been carefully exploited in [9,12]. Based on torus *CLOT* well implements the network locality forming the servers in close proximity to each other, which increases the communication efficiency. Besides, in response to the serious issue of long routing path length in torus topology, *CLOT* employs a number of low-end switches connecting the most distant node-pairs in each dimension, which largely shortens both network diameter and average path length; this in turn reduces the network latency. Meanwhile, *CLOT* also largely improves the bisection bandwidth, throughput and fault

[*] Corresponding author.
E-mail addresses: twangah@connect.ust.hk (T. Wang), wanglu.szu@qq.com, wanglu@szu.edu.cn (L. Wang), mhamdi@hbku.edu.qa (M. Hamdi).
[1] Student Member, IEEE
[2] Member,IEEE,
[3] Fellow, IEEE

https://doi.org/10.1016/j.comcom.2018.07.021
Received 5 November 2017; Received in revised form 17 June 2018; Accepted 18 July 2018
Available online 26 July 2018
0140-3664/ © 2018 Elsevier B.V. All rights reserved.

tolerance with better path diversity. Moreover, from the perspective of cost-effectiveness, *CLOT* also far outperforms other hierarchical topologies like FatTree [13] with regard to the network cost. Furthermore, the geographical address assignment mechanism enables content routing such as key-value stores in addition to traditional routing, and works well with legacy TCP/IP protocol. Besides, the proposed automatic address configuration mechanism and malfunction detection mechanism greatly facilitate the network configuration and avoids the human errors.

The rest of the paper is organized as follows. First we briefly review related works in Section 2. Section 3 gives the motivation of this research. Then Section 4 presents the design and analysis of *CLOT*. Afterwards, Section 5 discusses the network abstraction layers and the network address translation mechanism. Then Section 6 describes the automatic address configuration mechanism. Thereafter, the routing scheme and flow control designs are shown in Section 7. Section 8 presents the system evaluation and simulation results. Finally, Section 9 concludes this paper.

2. Related work

As a low cost, efficient and robust network architecture, torus fabric has received considerable research interest in recent years. Besides being widely used in the High Performance Computing (HPC), the torus-based interconnects have also recently been introduced to data center clusters, and the typical examples include CamCube [7], Small-World [8]. Fig. 1 gives an example of 1D&2D&3D Torus topologies.

CamCube [7] is a shipping container sized DCN based on a 3D torus topology, and it is the first work that introduces the torus fabric into data center interconnects. CamCube uses a traditional 3D torus topoloy (as shown in Fig. 1) without any changes to it. CamCube is one prototype that utilizes a low-level link oriented API to allow applications to implement their own particular routing protocols to optimize the application-level performance. These customized routing protocols run as services in the servers, which is similar to overlay network. The critical drawback is the long routing path length of torus based network, which results in low routing efficiency and high latency.

In order to decrease the average routing path length, researchers proposed a random data center topology named Small-World (SWDC) [8] which is built upon a regular pattern, such as ring, torus or tube. Small-World reduces the average routing path length by introducing a number of random links. The degree of each server is limited to six, where the number of random links of each node in SW-Ring, SW-2DTrous, and SW-3DHexTorus are 4, 2, and 1, respectively. However, the deterministic short-path based greedy routing algorithm results in low worst-case throughput and poor load balancing, which may lead to network congestion.

NovaCube [9] shares the similar goal of shortening routing path length, and improving the bisection bandwidth and throughput. Fig. 2 presents an example of 3D NovaCube. Based on *n*-D torus, NovaCube is constructed by adding a number of jump-over links in a beneficial way. Different from CamCube which is a 3D-Torus, NovaCube can be any dimensional Torus, with additional jump-over links. By doing this, NovaCube greatly reduces the network diameter, increases the bisection bandwidth, improves the worst-case throughput, and provides a

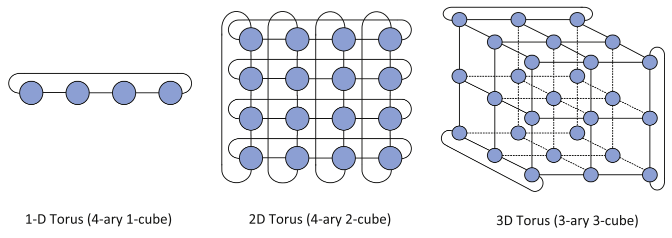

Fig. 1. Examples of 1D&2D&3D *Torus* topologies.

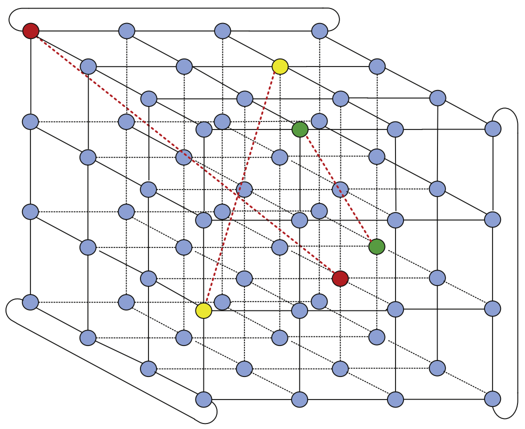

Fig. 2. An examples of 3D *NovaCube* topology (not allwrap-links are shown).

better fault tolerance as well. Additionally, its probabilistic weighted oblivious routing algorithm PORA helps NovaCube achieve good load balancing with near-optimal routing path length.

Different from CamCube, CLOT extended Torus (CamCube) topology by adding some most beneficial low-end switches (as shown in Fig. 4). Compared with NovaCube, CLOT employs similar methodology to improve the performance of Torus, where the differences are that CLOT introduces a certain number of switches while NovaCube adds some jump-over links between certain pair of server nodes. CLOT was firstly proposed in [12]. Based on this work, we enrich this paper by clearly stating the research motivation in Section 3, and in order to facilitate the network construction and management we extended this work by designing an automatic address configuration mechanism and an error detection mechanism in Section 6. Besides, we conducted new simulations in Section 8 to evaluate the CLOT from various aspects including average path length, throughput, latency, and fault tolerance.

3. Motivation

3.1. Why Torus Cluster

Torus is a switchless network interconnection without any switches, routers, or other network devices and associated cooling costs. The Torus-based architecture well implements the network *locality* forming the servers in close proximity of each other, which increases the communication efficiency. The multi-dimensional torus interconnection is attractive for a variety of reasons. Firstly, it results in lower infrastructure cost and energy cost because it does not need switches [14,15]. Even the wiring is simpler than other methods such as Fat-Tree. Secondly, it enjoys better reliability and fault-tolerance. The traditional architecture is usually constructed with a large number of switches. If in case one of the network devices fails, it will greatly impacts on the network performance and system reliability. For example, if a ToR switch fails, the whole rack of servers will lost the connection with the servers in other racks. Thirdly, the architectural symmetry of Torus topology optimizes the scalability and granularity of Clusters. It allows systems to economically scale to tens of thousands of servers, which is well beyond the capacity of fat tree switches. Fourthly, it can provide high network performance, which has been proven in high-performance systems and supercomputers, such as IBM's Blue Gene/L [16], Cray Gemini [17] and Blue Gene/Q (5D Torus) [18].

3.2. Issues of tours

Although torus topology holds many advantages, this design consistently suffers from poor routing efficiency compared to other designs, which was mainly due to the relatively long routing paths in torus networks. For a *k*-ary *n*-D torus network, its network diameter is as high as $\lfloor\frac{k}{2}\rfloor n$ hops. For example, in a 3D torus with radix 20, its maximum distance between two nodes is 30 hops. The long path length in turn results in relatively high latency. Besides, the traditional routing algorithm also undergoes various critical issues. For example, DOR (dimension-ordered routing) routing algorithm [19] leads to poor load balancing and low throughput, Valiant routing (VAL) routing algorithm [20] destroys locality and receives longer routing path length, and the adaptive routing algorithms like MIN AD [21] usually lead to non-optimal routing solutions, and are usually not deadlock free.

4. *CLOT* network structure

This section presents the design principle of *CLOT* architecture with comprehensive performance analysis from various aspects. Prior to introducing the physical interconnections, we first bring forward several theorems with proof, which provides the theoretical basis of *CLOT* design.

Theorem 3.1. *For any node $A(a_1, a_2, \dots, a_n)$ in a k-ary n-D torus (when k is even), assuming $\widehat{F}_A$=$B(b_1, b_2, \dots, b_n)$ is the farthest node from A, then B is unique and B's unique farthest node is exactly A.*

Proof. In a k-ary n-D torus, if B($b_1, b_2, \dots, b_n$) is the farthest node from A($a_1, a_2, \dots, a_n$), where $a_i \in [0, k)$, $b_i \in [0, k)$, then:

$$b_i = \left(a_i + \frac{k}{2}\right) mod\ k, \quad for\ \ \forall\ i \in [1, n] \tag{1}$$

Since the result of $(a_i + \frac{k}{2})\ mod\ k$ is unique, thus $\forall b_i \in\ [0, k)$ is unique. Hence, A's farthest node B is unique. Next, assume $\widehat{F}_B = A'(a_1', a_2', \dots, a_n')$, similarly we have:

$$a_i' = \left(b_i + \frac{k}{2}\right) mod\ k, \quad for\ \ \forall\ i \in [1, n] \tag{2}$$

By combining (1) and (2), we can get:

$$a_i' = \left\{\left[\left(a_i + \frac{k}{2}\right) mod\ k\right] + \frac{k}{2}\right\} mod\ k \tag{3}$$

$$\because a_i \in [0, k), \quad \therefore a_i + \frac{k}{2} \in \left[\frac{k}{2}, k + \frac{k}{2}\right) \tag{4}$$

(1) For the case of $a_i + \frac{k}{2} \in \left[\frac{k}{2}, k\right)$, we have

$$\begin{aligned} a_i' &= \left\{\left[\left(a_i + \frac{k}{2}\right) mod\ k\right] + \frac{k}{2}\right\} mod\ k \\ &= \left(a_i + \frac{k}{2} + \frac{k}{2}\right) mod\ k = (a_i + k)\ mod\ k = a_i \end{aligned} \tag{5}$$

(2) For the case of $a_i + \frac{k}{2} \in \left[k, k + \frac{k}{2}\right)$, we have

$$\begin{aligned} a_i' &= \left\{\left[\left(a_i + \frac{k}{2}\right) mod\ k\right] + \frac{k}{2}\right\} mod\ k \\ &= \left(a_i + \frac{k}{2} - k + \frac{k}{2}\right) mod\ k = a_i\ mod\ k = a_i \end{aligned} \tag{6}$$

As a consequence of the above, $a_i' = a_i$ for $\forall i \in [1, n]$. Therefore, $A'(a_1', a_2', \dots, a_n') = A(a_1, a_2, \dots, a_n)$, which means the farthest node from B is exactly A. This ends the proof. □

Lemma 3.1.1. *In an n-D Torus with radix k, for any node $A(a_1, a_2, \dots, a_n)$, its farthest node $\widehat{F}_A$= $((a_1 + \lfloor\frac{k}{2}\rfloor)\ mod\ k, (a_2 + \lfloor\frac{k}{2}\rfloor)\ mod\ k, \dots, (a_n + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$.*

Proof. For any node A($a_1, a_2, \dots, a_n$) in k-ary n-cube, its farthest nodes on the first dimension are $((a_1 + \lfloor\frac{k}{2}\rfloor)\ mod\ k, 0, \dots, 0), \dots, (0, \dots, (a_i + \lfloor\frac{k}{2}\rfloor)\ mod\ k, 0, \dots, 0), \dots, (0, \dots, 0, (a_n + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$. By analogy, A's farthest nodes on subsequent higher dimensions can be done in the same manner. For instance, its farthest nodes on the i-th dimension are to set i number of coordinates to be $(a_j + \lfloor\frac{k}{2}\rfloor)\ mod\ k$, where $j \in [1, n]$, and the rest are set to be 0. Consequently, there are $\binom{n}{i}$ number of farthest nodes on the i-th dimension. Therefore, A's farthest node in an n-D Torus with radix k is $\widehat{F}_A$= $((a_1 + \lfloor\frac{k}{2}\rfloor)\ mod\ k, (a_2 + \lfloor\frac{k}{2}\rfloor)\ mod\ k, \dots, (a_n + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$. This ends the proof. □

Theorem 3.2. *In a k-ary n-D Torus, for a certain node A, denoting its farthest nodes on the i-th dimension as $\{\widehat{F}_A^i\}$, then the total number of A's farthest nodes in all dimensions is $N = \sum_1^n |\{\widehat{F}_A^i\}| = 2^n - 1$.*

Proof. Derived from Theorem 3.2, we can get $\left|\{\widehat{F}_A^i\}\right| = \binom{n}{i}$. Thus, the total number of farthest nodes of node A in all dimensions can easily be computed as $N = \sum_1^n |\{\widehat{F}_A^i\}| = \sum_1^n \binom{n}{i} = 2^n - 1$, which ends the proof. □

4.1. Physical structure

4.1.1. Key principle

As aforementioned, Torus suffers a lot from its long network diameter, which results in communication inefficiency and extra latency. The key principle of *CLOT* construction is to reduce the routing path length and improve its overall network performance while retaining the Torus's merits. Based on regular torus topology, *CLOT* is established by adding a number of most beneficial small low-end switches. More specifically, in *CLOT* each node and its $2^n - 1$ farthest nodes in different dimensions are connected via a low-end switch. In order to better illustrate the specific architecture of *CLOT*, some representative cases are provided as below.

4.1.2. Cases

- **Case #1:** Take 2-D *CLOT* for example, for any node A, as indicated in Theorem 3.3, the number of its farthest nodes is three $(2^2 - 1)$ in total. Thus, each switch needs to connect any node and its three farthest nodes in total, where a 4-port low end switch is needed. The four nodes under the same switch are: (a_i, a_j), $((a_i + \lfloor\frac{k}{2}\rfloor)\ mod\ k, a_j)$, $(a_i, (a_j + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$, and $((a_i + \lfloor\frac{k}{2}\rfloor)\ mod\ k, (a_j + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$, respectively, where $i, j \in [1, k]$, and $a_i, a_j \in [0, k - 1]$. Fig. 3 (left) illustrates the architecture of a 2D *CLOT*.
- **Case #2:** Likewise, for the case of 3D *CLOT*, eight-port switches are needed. The eight nodes under the same switch are: (a_i, a_j, a_z), $((a_i + \lfloor\frac{k}{2}\rfloor)\ mod\ k, a_j, a_z)$, $(a_i, (a_j + \lfloor\frac{k}{2}\rfloor)\ mod\ k, a_z)$, $(a_i, a_j, (a_z + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$, $((a_i + \lfloor\frac{k}{2}\rfloor)\ mod\ k, (a_j + \lfloor\frac{k}{2}\rfloor)\ mod\ k, a_z)$, $((a_i + \lfloor\frac{k}{2}\rfloor)\ mod\ k, a_j, (a_z + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$, $(a_i, (a_j + \lfloor\frac{k}{2}\rfloor)\ mod\ k, (a_z + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$, and $((a_i + \lfloor\frac{k}{2}\rfloor)\ mod\ k, (a_j + \lfloor\frac{k}{2}\rfloor)\ mod\ k, (a_z + \lfloor\frac{k}{2}\rfloor)\ mod\ k)$, respectively, where $i, j, z \in [1, k]$, and $a_i, a_j, a_z \in [0, k - 1]$. Fig. 3 (right) gives an

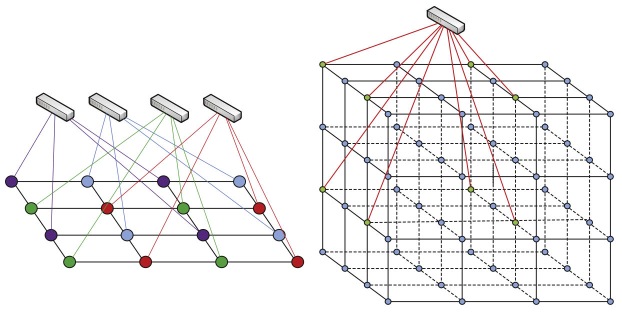
Fig. 3. Examples of 2D&3D *CLOT* topology (without wrap around links).

example of a 4*4*4 3D *CLOT*. For simplicity, the wrap around links are not displayed and only one switch is demonstrated.

- Generally, for a n-D *CLOT*, each switch connects to 2^n nodes: $((a_1 + i{*}\lfloor\frac{k}{2}\rfloor) \bmod k$, $(a_2 + j{*}\lfloor\frac{k}{2}\rfloor) \bmod k$, ... , $(a_n + r{*}\lfloor\frac{k}{2}\rfloor) \bmod k)$, where $i, j, r \in \{0, 1\}$.

4.2. Key properties and performance analysis

4.2.1. Network diameter

The length of routing path in a network greatly impacts the communication efficiency and transmission latency, while the network diameter largely determines the routing path length. Thus, a lower network diameter should be offered as a basic feature of network design. From this aspect, *CLOT* is highlighted by its low network diameter.

Theorem 3.3. *For a n-D CLOT with radix k, its network diameter is* $D_{CLOT} = \left\lfloor \frac{\left\lfloor \frac{k}{2} \right\rfloor + 1}{2} n \right\rfloor$.

Proof. As known, the network diameter of a regular k-ary n-Torus is $\lfloor\frac{k}{2}\rfloor n$, which means that starting from any node it can reach all the other nodes in the network within $\lfloor\frac{k}{2}\rfloor n$ hops. Assume S_i denotes the set of nodes that can be reached starting from A at the i-th hop, then the universal set of all nodes can be expressed as $S = \sum_{i=0}^{\lfloor\frac{k}{2}\rfloor n} S_i$. In *CLOT*, after connecting each node with its $2^n - 1$ farthest nodes in different dimensions through a switch, we define S_i' denotes the set of nodes that are i hops from node A, then there is: (1) when $\left\lfloor \frac{\left\lfloor \frac{k}{2} \right\rfloor + 1}{2} n \right\rfloor$ is even, we have $S_0' = S_0$, $S_1' = S_1$, $S_2' = S_2 + \sum_{i=1}^{n} S_{i*\lfloor\frac{k}{2}\rfloor}$, ... , $S'_{\frac{\lfloor \frac{k}{2} \rfloor + 1}{2} *n-1} = S_{\frac{\lfloor \frac{k}{2} \rfloor + 1}{2} *n-1} + S_{\frac{\lfloor \frac{k}{2} \rfloor + 1}{2} *n}$. (2) when $\left\lfloor \frac{\left\lfloor \frac{k}{2} \right\rfloor + 1}{2} n \right\rfloor$ is odd, we have $S_0' = S_0$, $S_1' = S_1$, $S_2' = S_2 + \sum_{i=1}^{n} S_{i*\lfloor\frac{k}{2}\rfloor}$, ... , $S'_{\left\lfloor \frac{\lfloor \frac{k}{2} \rfloor + 1}{2} n \right\rfloor -1} = S_{\left\lfloor \frac{\lfloor \frac{k}{2} \rfloor + 1}{2} n \right\rfloor -1} + S_{\left\lfloor \frac{\lfloor \frac{k}{2} \rfloor + 1}{2} n \right\rfloor +1}$, $S'_{\left\lfloor \frac{\lfloor \frac{k}{2} \rfloor + 1}{2} n \right\rfloor} = S_{\left\lfloor \frac{\lfloor \frac{k}{2} \rfloor + 1}{2} n \right\rfloor}$. It can be concluded that in *CLOT* any node A can reach all the other nodes within $\frac{\lfloor \frac{k}{2} \rfloor + 1}{2} n - 1$ or $\left\lfloor \frac{\lfloor \frac{k}{2} \rfloor + 1}{2} n \right\rfloor$ hops. Therefore, the network diameter of *CLOT* is $\left\lfloor \frac{\lfloor \frac{k}{2} \rfloor + 1}{2} n \right\rfloor$. □

4.2.2. Bisection bandwidth

If the network is segmented into two equally sized groups such that the bandwidth between these two groups is minimum, then bisection bandwidth is calculated as the sum of link capacities between the two halves. In a word, the bisection bandwidth represents the bandwidth across the ”narrowest” part of the network. It can be used to evaluate the worst-case network capacity [22].

Theorem 3.4. *The bisection bandwidth of n-D CLOT with radix k is:* $B_C = k^n + 4k^{n-1}$.

Proof. Assume *CLOT* network $N(N_1, N_2)$ is partitioned into two equal disjoint halves N_1 and N_2, and the set of links connecting two parts N_1, N_2 is denoted as $L(N_1, N_2)$. Each element of $L(N_1, N_2)$ is a bidirectional link with one node in N_1 and the other node in N_2. Thus, there are $|L(N_1, N_2)|$ bidirectional links or $2|L(N_1, N_2)|$ unidirectional channels at the bisection position. If the bandwidth of each unidirectional channel is set to be 1, then the bisection bandwidth of *CLOT* will be $B_C = 2|L(N_1, N_2)|$. For a k-ary n-D *CLOT* as depicted in Fig. 4, when k is even, there is even number of k k-ary $(n-1)$-D *CLOT*, which can be divided by the minimum bisection into two equal groups with $2k^{n-1}$ regular links and $\frac{k^n}{2}$ switch-server links. Therefore, there is $|L(N_1, N_2)| = 2k^{n-1} + \frac{k^n}{2}$. As a result, $B_C = 2|L(N_1, N_2)| = k^n + 4k^{n-1}$,

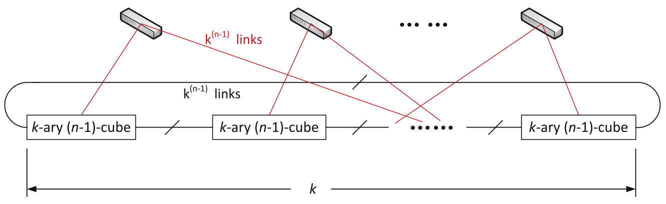

Fig. 4. An example of n-D *CLOT* with radix k.

which ends the proof. □

Compared with the traditional regular Torus network whose bisection bandwidth is $B_T = 4k^{n-1}$, *CLOT* effectively improves the bisection bandwidth by at least $\frac{B_C - B_T}{B_T} = \frac{k^n + 4k^{n-1} - 4k^{n-1}}{4k^{n-1}} = \frac{k}{4} \geq 25\%$, and the ratio increases accordingly as k increases. For instance, $k=10$ implies a 250% increment to the bisection bandwidth.

4.2.3. Throughput

The network throughput[4] is an important reference indicator to evaluate the network capacity of a topology. Throughput not only is limited by bisection bandwidth, but also largely depends on the traffic pattern, routing algorithm and flow control mechanism. However, we can evaluate the ideal throughput of a topology under the optimal routing and flow control. The maximum throughput occurs as some channel[5] in the network is saturated and the network cannot carry more traffic. Assume the workload on a channel c is γ_c, and the channel bandwidth is b_c, then the maximum channel workload of the network is $\gamma_{\max} = max\{\gamma_c, c \in C\}$. The ideal throughput Θ_{ideal} then can be obtained as:

$$\Theta_{ideal} = \frac{b_c}{\gamma_{max}} \tag{7}$$

Considering the uniform traffic pattern, the maximum channel load at the bisection channel has a lower bound which results in an upper bound on throughput. Because the *CLOT* is both node- and edge-symmetric, thus on average half of the k^n packets must cross through the B_C bisection channels. With assumption of the existence of optimal routing and flow control, the $\frac{k^n}{2}$ packets will be evenly distributed among all bisection channels which results in ideal throughput, and the lower bound of bisection channel load can be calculated as:

$$\gamma_{max} \geq \frac{\frac{k^n}{2}}{B_C} = \frac{\frac{k^n}{2}}{k^n + 4k^{n-1}} = \frac{k}{2k + 8} \tag{8}$$

Likewise, the upper bound of bisection channel load is

$$\gamma_{max} \leq \frac{k^n}{B_C} = \frac{k^n}{k^n + 4k^{n-1}} = \frac{k}{k + 4} \tag{9}$$

Combining Eqs. 7, 8 and 9, we can derive that

$$\frac{k+4}{k} b_c \leq \Theta_{ideal} = \frac{b_c}{\gamma_{max}} \leq \frac{2k+8}{k} b_c \tag{10}$$

Compared with the regular torus network whose ideal throughput is only $[\frac{4b_c}{k}, \frac{8b_c}{k}]$ [23], *CLOT* exhibits a considerable improvement to the network capacity regarding to the maximum throughput. From another perspective, if the throughput under a certain routing algorithm is normalized to the network capacity $\widehat{\Theta}_n = \frac{\gamma_{\max}}{\gamma_n(\widehat{R})}$, where $\widehat{R}$ denotes a routing algorithm, then in torus its maximum normalized throughput $\widehat{\Theta}_n$ can reach as high as 50% under a throughput optimal routing algorithm like Valiant routing (VAL) [20]. Consequently, in *CLOT* the

[4] The throughput means the *accepted traffic*, and it is the rate that traffic (bits/s) is delivered to the destination nodes.

[5] The channel means one path consisting of a series of links between a pair of nodes.

maximum normalized worst-case throughput can be achieved by at least 62.5%, which which is further evidence of *CLOT*'s better performance.

4.2.4. Path diversity

CLOT is vastly superior to other DCN architectures in path diversity. The number of distinct paths existing in *CLOT* is too huge to be calculated exactly, for simplicity, we first consider only the equal-cost shortest paths with the same direction in a regular torus without considering the switch-server links. Assume two nodes $A(a_1, a_2, ..., a_n)$ and $B(b_1, b_2, ..., b_n)$ where A and B are separated by $\Delta_i = |a_i - b_i|$ hops in the i-th dimension, then the total number of minimal paths N_{AB} between A and B is given by:

$$N_{AB} = \prod_{i=1}^{n} \binom{\sum_{j=i}^{n} \Delta_j}{\Delta_i} = \frac{(\sum_{i=1}^{n} \Delta_i)!}{\prod_{i=1}^{n} \Delta_i!} \tag{11}$$

where the term $\binom{\sum_{j=i}^{n} \Delta_j}{\Delta_i}$ computes the number of ways to choose where to take the Δ_i hops out of the remaining $\sum_{j=i}^{n} \Delta_j$ hops. For example, given $n = 3$, $\Delta_x = 4$, $\Delta_y = 5$, $\Delta_z = 6$, the total number of minimal paths N_{AB} is as high as 630630, and it increases rapidly with dimension. Besides, if taking the switch-server links into consideration the number of possible paths will be much larger, and the number of non-minimal possible is nearly unlimited. The good path diversity of *CLOT* offers many benefits to the network. On the one hand, the traffic can be selectively distributed over these equal-cost paths which can help achieve good load balancing. On the other hand, this path diversity also provides good fault tolerance where the traffic can route around the faulty channels by taking alternative equal-cost paths.

4.2.5. Cost-effectiveness

Apart from its strong advantages in network performance, *CLOT* is also very cost-effective.

It can be easily derived that in a k-ary n-D *CLOT*, the total number of switches is $N_{switch} = \left(\frac{k}{2}\right)^n$ with 2^n ports on each switch.

As known, the per-port-price of a switch increases with its number of ports. Based on careful investigations and statistics on the Amazon [24] online quoted prices, Table 1 gives the average per-port-price for 10GE switches with different numbers of ports. Table 2 exhibits the comparison of switch cost between different sized 3D *CLOT*s and Fat-Tree networks. It can be seen that a 3D *CLOT* uses only low-end 8-port cheap switches, while Fat-Tree requires high-end expensive switches with more ports. Although *CLOT* uses more switches than FatTree when the network size is larger than 16,000 servers, *CLOT* still yields a much lower switch cost. For example, FatTree requires an expenditure of 650,049,875$ on switches to scale up to 200,000 servers while *CLOT* only needs 375,000$ for the same network size which is around 173 times cheaper, which demonstrates the cost-effectiveness of *CLOT*.

5. Network layering and address translation

5.1. Network layering

Similar to the Internet protocol suite, *CLOT* network also uses encapsulation to provide abstraction of protocols and services. As illustrated in Fig. 5, the layered protocol stack in *CLOT* network is also

Table 1
The per-port cost on different types of switches.

Number of ports	Price per port (US $)
16	150
16–32	450
>32	650

Table 2
The cost comparison between Fat-Tree and 3-D *CLOT*.

Network size	Switch number (*p*-port switch)		Switch cost (US $)	
	Fat-Tree	*CLOT*	Fat-Tree	*CLOT*
500	199 (p=13)	63 (p=8)	376,086	9,450
1,000	315 (p=16)	125 (p=8)	750,047	18,750
5,000	922 (p=27)	625 (p=8)	11,262,119	93,750
10,000	1463 (p=34)	1250 (p=8)	32,522,033	187,500
16,000	2000 (p=40)	2000 (p=8)	52,000,000	300,000
20,000	2321 (p=43)	2500 (p=8)	65,005,758	375,000
100,000	6787 (p=74)	12,500 (p=8)	325,045,783	1,875,000
200,000	10,773 (p=93)	25,000 (p=8)	650,049,875	3,750,000

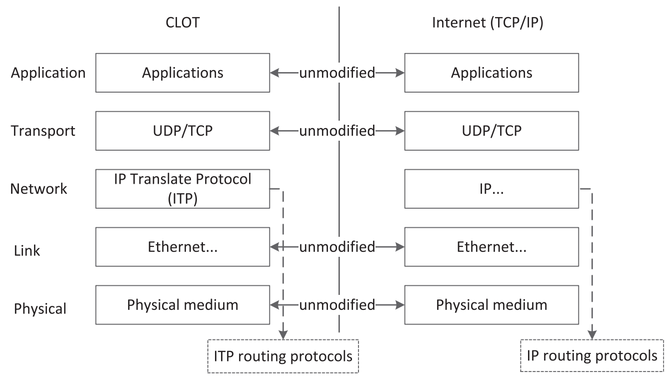

Fig. 5. The *CLOT* network abstraction layers.

divided into five layers (application, transport, network, link and physical), which enables an application to use a set of protocols to send its data down the layers, being further encapsulated at each level.

The major difference lies in the network layer, where the traditional Internet uses an IP address to locate different hosts while *CLOT* uses coordinates to direct the data transmission with the benefit of the perfect symmetry of topology, which facilitates the routing greatly. Coupled with the geographical identities, the geographical routing and addressing APIs make it possible for *CLOT* to implement applications like key-value stores, map-reduce operation and coordination protocols more efficiently. In order to keep the running applications unmodified, *CLOT* makes no changes to the transport protocols which enables the applications to directly adopt the functionality of TCP or UDP. Since the IP address must be provided when creating sockets for TCP and UDP, however, the *CLOT* only has coordinates, so the network needs an adaptation layer to translate coordinates to the IP address format.

5.2. Network address translation

In order to make the *CLOT* network compatible with the legacy TCP/IP protocol, we designed an address translation mechanism to implement the convention between IPv4 address and *CLOT* coordinates.

As illustrated in Fig. 6, a 32-bit IPv4 address is divided into seven segments including six pieces with five bits and one piece with two bits. The coordinate of each dimension is denoted by the five-bit slice, and the remaining two bits are a dimension flag which is used to indicate the number of dimensions. In this way, a 32-bit IPv4 address can support up to six dimensions, where a 6-D *CLOT* can hold up to 2^{30} = 1,073,741,824 (1 billion) servers, thus this kind of division is reasonable and adequate even for a large scale data center. However, the two-bit dimension flag can not represent six dimensions. Here we define that only the dimension flag with "11" indicates a 6D network address. When the number of dimensions is less than six, the address space of last dimension will not be used. Therefore, when the dimension flag is "10", we make use of the first three bits of the last dimension's address space to represent the specific dimension. The rule of dimension

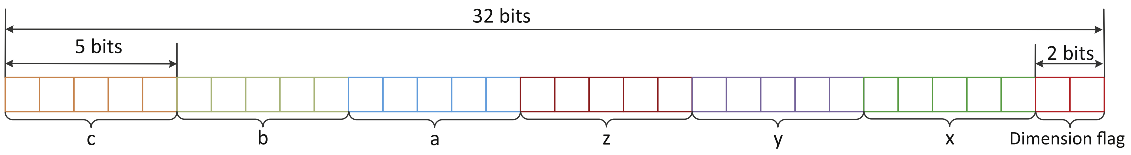

Fig. 6. The correspondence between IPv4 and *CLOT* coordinates.

Table 3
The representation of different specific dimensions.

Dimension flag (binary)	First 3 Bits of *c* (binary)	Dimension number (decimal)
11	xxx	6
10	101	5
10	100	4
10	011	3
10	010	2
10	001	1

correspondence is illustrated in Table 3, and other values are currently considered illegal.

6. Automatic address configuration mechanism design

6.1. Motivation

A multiple dimensional torus network can easily scale to support a large number of servers. It will turn to be very complex to manually configure the network address for each node, and it possibly incurs human errors which are difficult to be correctly detected. Besides, node failures, network interface card error, and wrong link connections also greatly affect the network operation. Therefore, the system requires an automatic network address configuration mechanism and an efficient error detection mechanism to facilitate the network construction and management.

6.2. Automatic address configuration mechanism

In this subsection, the Automatic Address Configuration Mechanism (AACM) is designed. Originally, as all nodes have no addresses before executing AACM, thus the AACM needs to work on the link layer other than network layer to distribute the configured addresses. As aforementioned, in *CLOT*, the node addresses are denoted by coordinates, and the coordinates are linearly increased on each dimension.

6.2.1. Address configuration frame format

In *CLOT* network, the link layer uses point-to-point connection, and each port of one node only can directly send packets to its directly-connected port of its next neighbor node, without need of MAC addresses. Therefore, the data frame only needs to carry the protocol type of upper layer and no need to contain MAC addresses, as illustrated in Fig. 7. For example, the *Protocol* field with the value of 0x0800 denotes that the upper layer is IPv4, and 0x86dd denotes IPv6. When a node receives a frame, it analyzes the *Protocol* field of this frame and push it to the corresponding protocol instance for further processing. Here, we set the *Protocol* field as 0xA800 to denote the automatic address configuration protocol.

The format of frame used in the address configuration protocol is

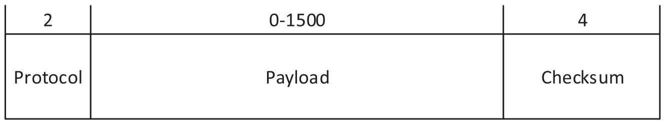

Fig. 7. Frame format.

illustrated in Fig. 8. Once a node identifies the *Protocol* field of its received frame is 0xA800, then this frame will be handled by address configuration protocol. The *Flag* field contains 8 bits. The *Config* bit with value ”1” indicates this frame is used to configure the node coordinate using the value contained in *Address* field. The *Alt* bit is used together with *Config* bit. The *Alt* bit will be inversed between two consistent configuration, so that the node can identify it as a new configuration other than an address configuration error. The *ACK* bit denotes an confirmation frame, where each node should send back an ACK frame to confirm its correct configuration status. Setting the *Addr Err* bit indicates that there are address configuration errors in the network. The *Dead link* bit with value ”1” means some node is not reachable due to node/link failure. It is usually used together with *Report* bit. The *Success* bit indicates a successful address configuration. The *Report* bit with value ”1” denotes the frame should be routed back to the origin node. Finally, setting the *Combine* bit can reset the other bits and be used in combination with other bits to implement more functions. This is used for future protocol extension.

The 32-bit *Address* field contains the address to be configured. As denoted in Table 3 and Fig. 6, the *Address* also contains the dimension information, thus each node can learn the dimension of the network. The next field *Node number list* indicates the number of nodes on each dimension in the network, so that each node can compute if it is located at the edge of network, where the node is connected with its neighbor node by a wraparound link. The variable content of *Data* field depends on the settings of *Flags* field. For example, in an error report frame with *Addr Err* =1 and *Report* = 1, the *Data* field will contain the address/location information of error node or link.

6.2.2. Auto address configuration algorithm

Algorithm 1 details the working procedure of the automatic address configuration algorithm. To better explain the principle of the automatic address configuration and error detection mechanism, Fig. 9 depicts an AACM example in a 2D *CLOT* network. In torus topology each node is equal and thus AACM algorithm can be executed starting from any node. Assume the starting origin node is the upper left corner node, with the coordinate (0, 0). The node identifies itself as an origin node (0, 0), then it sends configuration messages to its neighbor nodes along +x, -y, -x, +y directions (drawn as red lines in Fig. 9): (0, 1), (1, 0), (0, 3), and (3, 0), respectively. If there are no errors, all nodes that receive configuration messages should reply an ACK message to confirm the configurations, and meanwhile send configuration messages to their next hop nodes. For example, when node (0, 1) receives the configuration message and checks the y coordinate is 0, then it will not wait the confirmation messages from y directions and directly send configuration messages to its neighbor nodes along +x, +y, -y with configured addresses (0, 2), (1, 1) and (3, 1), respectively, and wait for confirmation ACKs. Likewise, node (1, 0) takes similar actions. These actions are drawn in green lines as shown in Fig. 9. When node (1, 1) receives an address configuration request from one direction (saying -y), and it checks all coordinates are non-zero values and not maximum values (i.e. edge node), thus it must wait the configuration request from another direction (i.e. -x). When it receives the configuration message from -x direction, this node compares two received addresses from directions -y and -x. If they are both (1, 1), then it considers the configured addresses of previous nodes are all correct, and then sends address configuration requests (1, 2) and (2, 1) to its next hop nodes along +x and -y directions, respectively, as shown in Fig. 9 (drawn in blue lines).

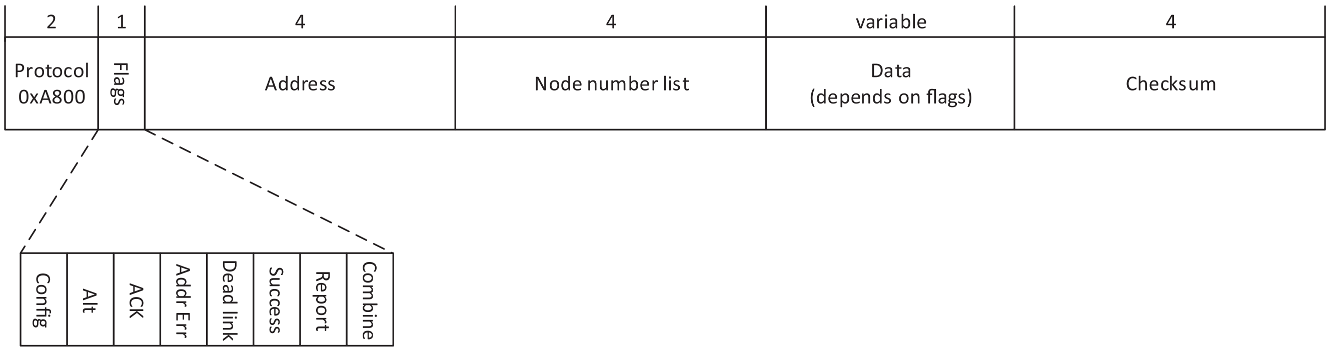

Fig. 8. AACM frame format.

```
function ADDRESS_CONFIGURATION
    while Node receives address configuration request do
        $n \leftarrow$ gets the number of dimensions
        $(x_0, x_1, \ldots, x_{n-1}) \leftarrow$ gets coordinates
        $(k_0, k_1, \ldots, k_{n-1}) \leftarrow$ gets the number of nodes on each dimension
        Reply ACK message
        if $x_i$==0 for $\forall i \in [0, n-1]$ then
            Set current address as origin
            Send corresponding configuration requests from all ports, and wait ACK msg.
        elseif$\exists i x_i$ != 0
            if $\forall i \in [0, n-1]$, $x_i < n-1$ then
                if All the non-zero coordinate's negative direction has received config requests then
                    if All requested addr are the same then
                        Configure with the requested addr
                        Send config. requests to all non-zero coordinates' positive directions, and wait for ACK msg.
                        Send config. requests to zero coordinates' both positive and negative directions, and wait for ACK msg.
                    elseSend erro msg from the port which firstly receives the config. request.
                    end if
                end if
            else
                if Non-zero coordinates' negative direction and max coordinates' positive direction have received config. requests then
                    if All requested addrs are the same then
                        Configure with the requested addr
                        Send config. requests to all non-zero coordinates' positive directions, and wait for ACK msg.
                        Send config. requests to zero coordinates' both positive and negative directions, and wait for ACK msg.
                    elseSend erro msg from the port which firstly receives the config. request.
                    end if
                    if $x_0 = k_0$ && … && $x_{n-1} = k_{n-1}$ then
                        if all ports have received configuration requests with the same addr then
                            Send configuration success msg to the port firstly receiving the config. request
                        end if
                    end if
                end if
            end if
        end if
    end while

    while the node waiting config. request times out do
        Send node unreachable erro msg from the port which firstly receives the config. request.
    end while
end function
```

Algorithm 1. Automatic address configuration.

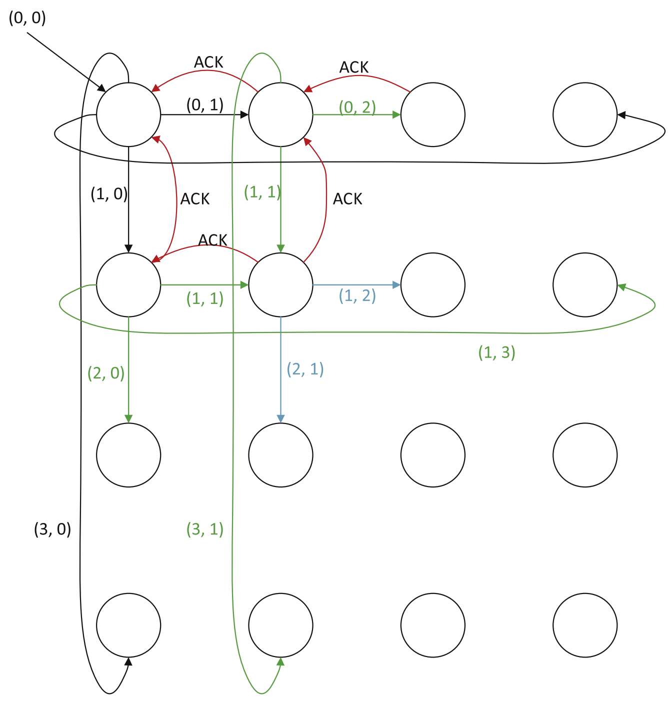

Fig. 9. An example of AACM.

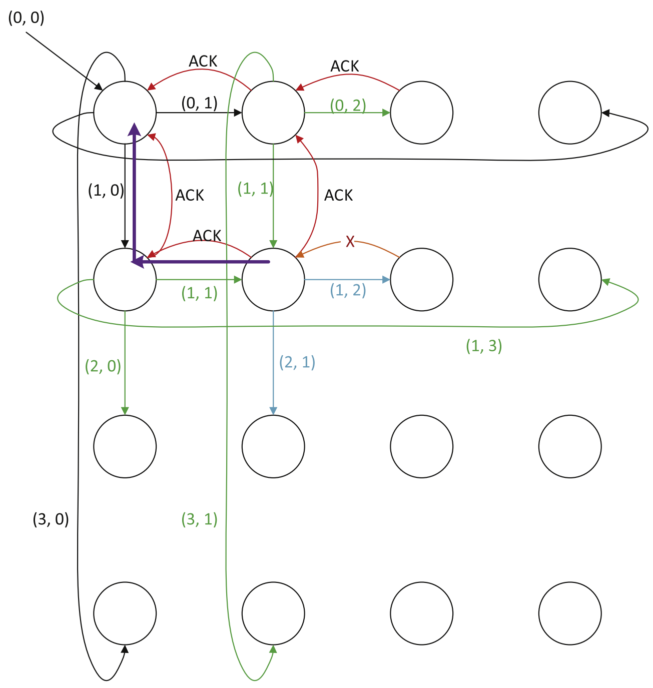

Fig. 10. AACM in case of link/node error.

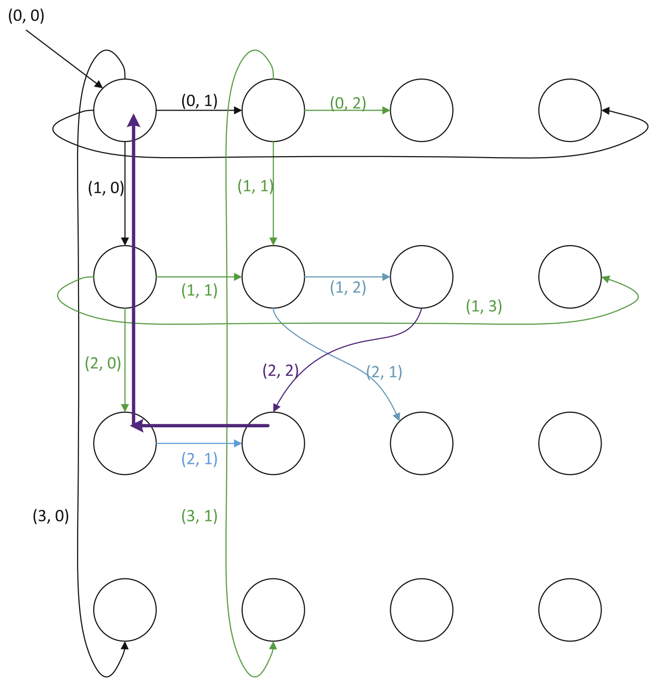

Fig. 11. AACM in case of human error.

As the address configuration carries on, if there are no errors, then the last node (bottom right corner node) of the network will receives the same address configuration request from all directions. Then, the node identifies itself as the last node (all coordinates have maximum values) of the whole network, and sends successful configuration report to the origin node (0, 0). Notably, the whole configuration procedure is executed in parallel.

The above is an error-free case with successful configurations. However, sometimes network errors may occur in practice, such as unreachable nodes, wrong connections caused by human errors. The AACM will automatically detect these errors and explicitly report the error reasons and error locations to the origin node. As demonstrated in Fig. 10, suppose node (1, 1) sends an address configuration request to node (1, 2), however, due to some errors (e.g. link disconnection, port error, node error, etc.) occurred in node (1, 2), node (1, 1) could not receive confirmed ACK message from node (1, 2). After waiting a certain time, when the node (1, 1)'s predetermined timer timeout, an error report containing the coordinates of node (1, 1) and node (1, 2) will be generated and sent back to the origin node via DOR routing. It is notable that the node (1, 1) being able to send address configuration request to node (1, 2) means that the previous nodes of node (1, 1) in each dimension have been configured successfully, therefore, the error report can be guaranteed to be routed back to the origin node to inform the administrator about the errors.

Another error case is the wrong link connection caused by human errors, as shown in Fig. 11. In this case, only using ACK confirmation messages cannot detect such errors. Nevertheless, AACM can efficiently detect this error by checking the consistency of received address configuration requests from all directions. After error being detected, the error node will not send address configuration requests to its next hop nodes so as to prevent subsequent nodes generating numerous error reports. As shown in Fig. 11, because of wrong connections caused human errors, node (2, 1) will receive two different address configuration requests from two dimensions, one correct request with address (2, 1) sent from node (2, 0) and one wrong request with address (2, 2) sent from node (1, 2). The inconsistency of two requests invokes node (2, 1) to send error report to the port which firstly receives the configuration request. Then the error report, containing the node's coordinate information, the port number that receives the error report and all inconsistent coordinates of error nodes, will be forwarded to the origin node along the DOR routing path.

7. Routing scheme and flow control

This section presents a Probabilistic Oblivious Weighted routing algorithm named POW and flow control mechanism for *CLOT*, which aim to achieve good load balancing with minimum routing path and high throughput.

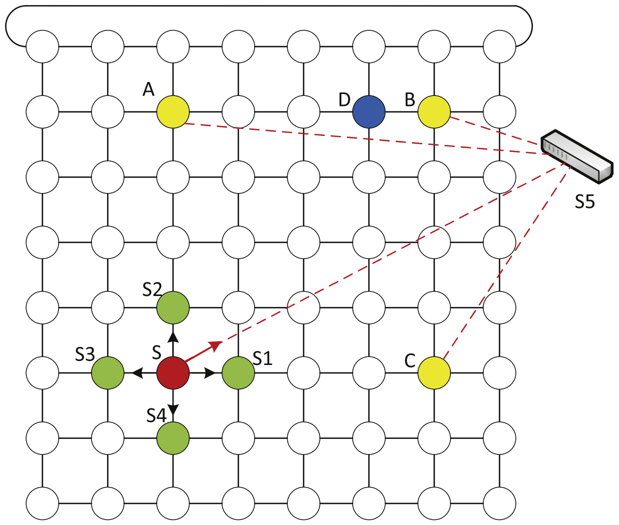

Fig. 12. POW routing in an 8*8 2-D *CLOT* (for simplicity not all switches and wrap-around links are shown).

7.1. POW routing algorithm

Different from the deterministic routing algorithms (e.g. DOR [19]) whose routing is directly determined by the source and destination address eliminating path diversities provided by torus topology incurring poor load balancing or even congestion, POW makes a probabilistic decision at each routing step according to a distance-based probability function. Here we use the notation Δ_{SD} to denote the DOR [19] distance (without switches) between node S and D.

Without loss of generality, for simplicity we use 2D *CLOT* to illustrate the working procedures of POW algorithm. As shown in Fig. 12, assume a source node S needs to send data to a destination node D, then S has five directions in its first step to route the data which are S_1, S_2, S_3, S_4 and S_5, where S_5 is a switch node. In order to select the most beneficial next-hop node, each direction is assigned a probability based on the distances between each next-hop node and destination node. Then POW decides the next-hop node according to their probabilities. In order to guarantee the packet delivery, the distance Δ_{S_iD} between the chosen node S_i and destination D must be smaller than Δ_{SD}, i.e. $\Delta_{S_iD} < \Delta_{SD}$. The normalized probability function is computed as below:

$$p_i = \frac{\frac{1}{\Delta_i^2}}{\sum_{i=1}^{\psi} \frac{1}{\Delta_i^2}} \tag{12}$$

where ψ is the number of satisfied neighbours of the source node. For example, in Fig. 12 we have $\Delta_{SD} = 7$, and the distances between S's neighbour nodes and destination D are $\Delta_{S_1D} = 6$, $\Delta_{S_2D} = 6$, $\Delta_{S_3D} = 8$, $\Delta_{S_4D} = 6$, $\Delta_{S_5D} = 2$, respectively, where S_3 fails to meet the requirement of $\Delta_{S_iD} < \Delta_{SD}$. Thus, the probability of choosing S_1 as the next-hop is $p_{S_1} = \frac{\frac{1}{\Delta_{S_1D}^2}}{\sum \frac{1}{\Delta_{S_iD}^2}} = 8.33\%$ $(i = 1,2,4,5)$, and likewise $p_{S_2} = 8.33\%$, $p_{S_4} = 8.33\%$, and $p_{S_5} = 75.00\%$, respectively. Clearly, POW prefers to choose the shorter route with a higher probability. Here we have two individual cases to deal with: If the switch S_5 is chosen finally, then it will determinedly choose one of its neighbour nodes that is closest to the destination as the next-hop. Otherwise, if a normal node S_i, $i = 1,2,4$, is selected as S's next-hop, then the above procedure is repeated taking S_i as the new source node until the packet reaches the destination.

7.2. Flow control

The flow control mechanism is designed to manage the allocation of resources (e.g. queue buffer) to packets as they progress along their route so as to avoid congestion or deadlock. In the traditional Internet, protocols with flow control like TCP usually avoid congestion through dropping packets and retransmission. However, this mechanism is not suitable in the torus network due to its long routing path, e.g., if the hot spot is very far from the source then dropping packet means the previous long transmission would have been of waste and the retransmission packets would occupy additional bandwidth which may make the network more congested. Thus, the most beneficial way is to achieve a packet lossless fabric, where the buffers of nodes in the network should be guaranteed no overflow. The common approach is to use credit-based flow control. As shown in Fig. 13, the prerequisite for any uplink node sending data to a downlink node is that its output port has enough credits. The default value of credit is the number of packets that the downlink node can accept. The credits will be deducted by one whenever one packet is sent out. Correspondingly, whenever the downlink node vacates new buffer space to accept a new packet, it will send one credit to its uplink node and increase the credit of its corresponding port by one. Usually, there may exist a nearly negligible delay between packet transmission and credit feedback between two directly connected nodes, however, for safety the downlink node should have slightly larger buffer for several more packets to deal with any possible underflow issues. The credit can either be transferred via in-band or out-of-band signal, where the in-band way is more cost-effective but complicated to be implemented while out-of-band method is simple to be realized but with more cost. Besides, the Head-of-Line blocking issue can be addressed in the way of shared buffering where the output queue and input queue are logically organized as a shared virtual queue in the implementation.

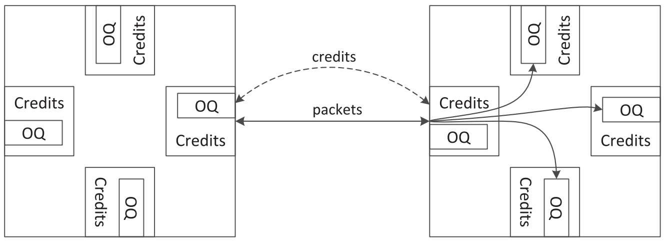

Fig. 13. Credit-based flow control implementation.

7.3. Deadlock avoidance implementation

Deadlock occurs when a cycle of packets are waiting for one another to release resources (queue buffer) where the resource dependencies form a cycle. Packets are blocked indefinitely and throughput will also collapse at once. There are usually two deadlock avoiding techniques, including virtual channels (by decomposing each unidirectional physical channel into several logical channels with private buffer resources) and Turn Model Routing (by eliminating certain turns in some dimensions). To make our architecture deadlock free, we first ensure that the route cannot form a loop in POW routing. As aforementioned, each step of POW routing is guaranteed to be closer to the destination without jumping back, which ensures POW is deadlock free in the mesh sub-network. Finally, if packets go through the wraparound links in torus network, then as was done in [19] we use two virtual channels to cut the loop into two different logical channels, which is easy to be implemented and effectively avoids deadlock.

7.4. Livelock prevention

Livelock is another notorious issue where packets continue to proceed through the network but do not advance towards their destination even though they are not blocked. This may occur when non-minimal adaptive routing is allowed where packets may be misrouted but are not able to get closer to their destination. In POW routing, it guarantees the packet delivery to destination in a greedy way, where the decision made at each step is based on the distance which requires $\Delta_{S_iD} < \Delta_{SD}$. Thus, POW routing enables packets to choose its next hop which is always closer to the destination than that from the current node, and with a high probability of choosing the shortest path. Therefore, we can claim that POW is a livelock-free routing algorithm.

8. Evaluation

This section presents the evaluation results of *CLOT* and POW routing algorithm from various aspects by using network simulator 3 (NS-3) [25].

8.1. Simulation settings

In this section, all the simulations are conducted by applying the All-to-All traffic pattern. The packet inter-arrival time reveals an ON/OFF pattern. Its distribution follows the Lognormal mode for OFF phase, and for ON phase the Exponential Flow Mode is applied to determine the distribution of packet inter-arrival times. All the links are capable of bidirectional communications, and the link bandwidth is set to be 1 GBps. The default MTU of a link is 1500 bytes, the default packet size is one MTU, and the default buffer size of each switch is 10 MTU. The default processing time for a packet at a node is 10 μs while the default propagation delay of a link is 5 μs, and the TTL of a packet is set to be 128. In faulty conditions, the failure rate will be set fixed and the fault links will be chosen at random.

8.2. Average path length

The average path length (APL) and network diameter largely determine the network latency. A smaller network diameter with lower average path length should be offered as a desired feature of the network design enabling the data center to provide faster services. Fig. 14 illustrates the simulation results of average path length and network diameter achieved by *CLOT* and its competitors NovaCube and CamCube, applying all-to-all traffic pattern. It can be seen that *CLOT* reduces the network diameter by 35% ~ 50% comparing with CamCube for different network sizes. Moreover, *CLOT* achieves smallest average path length, which is 10% ~ 20% smaller than NovaCube and 40% ~ 50% smaller than CamCube. The smaller average path length will help *CLOT* achieve a lower network latency, which is further convinced by the simulation results in Section 8.4.

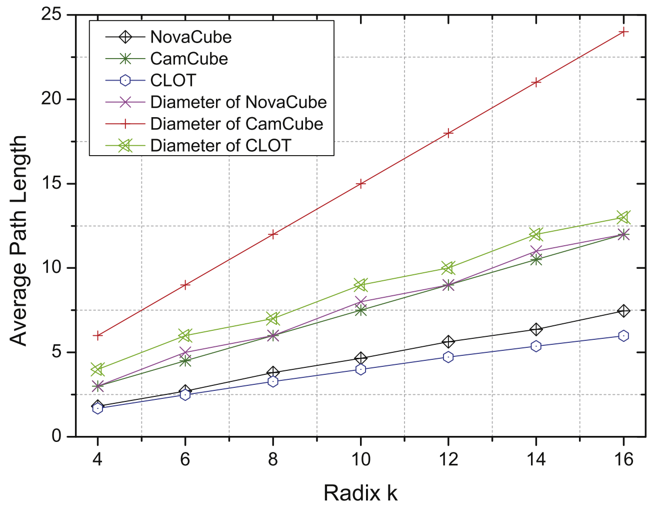

Fig. 14. The performance of average path length.

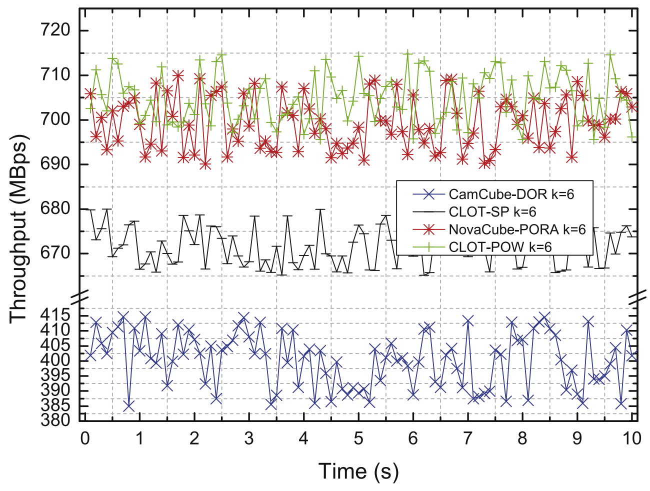

Fig. 15. The performance of throughput for 216-server network.

8.3. Throughput

The throughput is evaluated to measure the overall network capacity of the architecture. The throughput is usually limited by bisection bandwidth and also impacted by the routing algorithm. In the simulations, NovaCube applies PORA routing algorithm, CamCube uses DOR (dimension-ordered routing) routing algorithm, and *CLOT* adopts shortest path (SP) routing and its POW routing algorithm. Fig. 15 and Fig. 16 exhibit the evaluation results of throughput with different network configurations, where the evaluated network sizes are 216 servers and 512 servers, respectively. The results reveal that for 216-sized network *CLOT* improves the throughput by 84.22% at most and 76.13% on average comparing with regular torus based CamCube. The throughput improvement for 512-sized network is higher, where the increasement is 94.38% at most and 82.73% on average comparing with CamCube. This proves the previous theoretical analysis of *CLOT*'s bisection bandwidth and throughput. Besides, *CLOT* and NovaCube achieve a similar performance on throughput, where *CLOT* is slightly better than NovaCube with a 3% ~ 4% improvement on average.

8.4. Latency

The network latency in the simulations is measured using the average RTT/2 (round-trip-time) of ICMP packets between two designated hosts by repeating 10 times. The latency t is calculated as below:

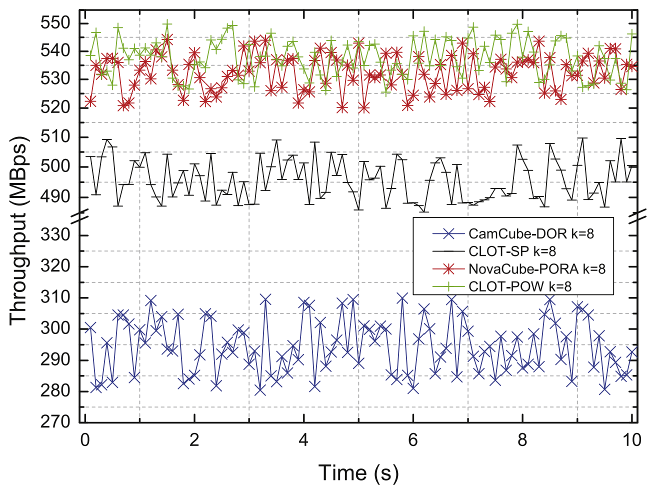

Fig. 16. The performance of throughput for 512-server network.

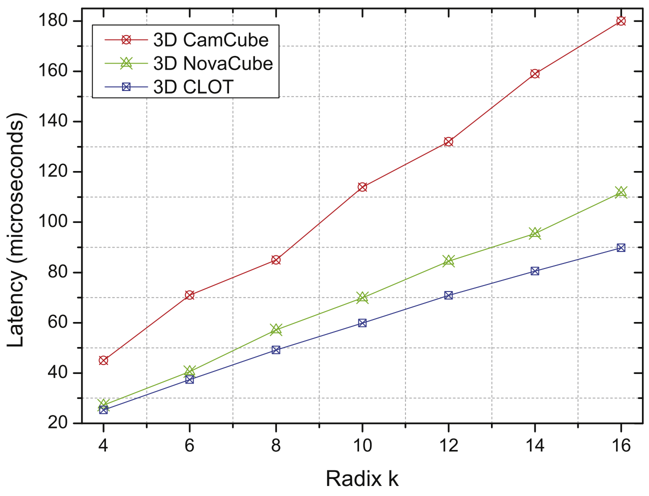

Fig. 17. The performance of network latency with different network configurations.

$$t = \sum_{i=1}^{N} \frac{RTT_i}{2}/N \tag{13}$$

where $N = 10$ indicates the repeating times, and RTT_i denotes the round trip time of i-th ICMP packet.

Fig. 17 shows the simulation results of network latency of CamCube, NovaCube and *CLOT* with different network configurations varying from k = 4 (64 servers) to k = 16 (4096 servers). The results show that *CLOT* achieves lowest latency, which is 42.2% ~ 50.1% lower than CamCube and 7.3% ~ 19.7% lower than NovaCube. It also reveals that CLOT will gain more latency reduction as the network size increases (with higher k).

8.5. APL under faulty conditions

As aforementioned in Section 4.2.4, torus-based *CLOT* architecture achieves very good fault tolerance benefiting from its excellent path diversity and rarely confronts network disconnections. Fig. 18 illustrates the results of the achieved average path length (APL) under faulty conditions with different link/node failure ratios. Noting that the statistic result excludes the cases of connection failures between two given servers (whose APL is infinite), only calculating the average path length of reachable server pairs. From the results shown in Fig. 18 it can be seen that apparently the average path length increases as link/node failure ratio increases for both *CLOT* and CamCube. Nevertheless, *CLOT* achieves a better performance than CamCube under faulty conditions, where the average path length of CLOT is 42% ~ 50% lower on overage than that of CamCube. Another interesting finding is that the node failure has a slightly higher impact on the average path length.

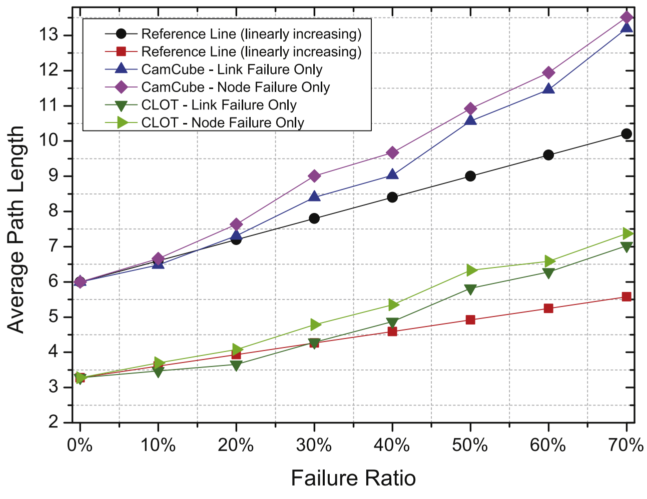

Fig. 18. The performance of fault tolerance.

9. Conclusion

This paper presents a novel torus based DCN architecture *CLOT* using a certain number of low-end switches to connect the most distant nodes in each dimension. Comparing with the regular k-ary n-D torus network *CLOT* halves the network diameter, improves bisection bandwidth by at least 25% where the ratio increases with k, increases the ideal throughput to be at least 65%, and provides a much better path diversity resulting in a better fault tolerance. Additionally, coupled with the geographical coordinate address, *CLOT* can carry out the content routing which provides the possibility of implementing key-value stores in addition to the traditional routings. *CLOT* is also compatible with legacy TCP/IP protocols. The automatic address configuration mechanism and error detection mechanism significantly facilitate the network construction and management. Furthermore, under the specific flow control mechanism the livelock/deadlock free probabilistic weighted routing algorithm POW helps *CLOT* achieve a better load balancing than the deterministic routing algorithms. The extensive simulations further prove the good performance of *CLOT* network with respect to average path length, throughput, network latency and fault tolerance.

Acknowledgement

The research was supported in part by the China NSFC 61502313, Shenzhen Science and Technology Foundation JCYJ20170817095418831, Tencent”Rhinoceros Birds” – Scientific Research Foundation for Young Teachers of Shenzhen University. Lu WANG is the corresponding author.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comcom.2018.07.021.

References

[1] T. Wang, Z. Su, Y. Xia, M. Hamdi, Rethinking the data center networking: architecture, network protocols, and resource sharing, Access, IEEE 2 (2014) 1481–1496.
[2] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, Bcube: a high performance, server-centric network architecture for modular data centers, ACM SIGCOMM Comp. Commun. Rev. 39 (4) (2009) 63–74.
[3] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, S. Lu, DCell: A scalable and fault-tolerant network structure for data centers, ACM SIGCOMM Comp. Commun. Rev. 38 (4) (2008) 75–86.
[4] T. Wang, Z. Su, Y. Xia, Y. Liu, J. Muppala, M. Hamdi, Sprintnet: A high performance server-centric network architecture for data centers, Communications (ICC), 2014 IEEE International Conference on, IEEE, 2014, pp. 4005–4010.
[5] T. Wang, Z. Su, Y. Xia, J. Muppala, M. Hamdi, Designing efficient high performance server-centric data center network architecture, Comput. Networks 79 (2015) 283–296.
[6] D. Lin, Y. Liu, M. Hamdi, J. Muppala, Flatnet: Towards a flatter data center network, Global Communications Conference (GLOBECOM), 2012 IEEE, IEEE, 2012, pp. 2499–2504.
[7] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O’Shea, A. Donnelly, Symbiotic routing in future data centers, ACM SIGCOMM Comp. Commun. Rev. 40 (4) (2010) 51–62.
[8] J.-Y. Shin, B. Wong, E.G. Sirer, Small-world datacenters, Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM, 2011, p. 2.
[9] T. Wang, Z. Su, Y. Xia, B. Qin, M. Hamdi, Novacube: A low latency torus-based network architecture for data centers, Global Communications Conference (GLOBECOM), 2014 IEEE, IEEE, 2014, pp. 2252–2257.
[10] T. Wang, Y. Xia, D. Lin, M. Hamdi, Improving the efficiency of server-centric data center network architectures, Communications (ICC), 2014 IEEE International Conference on, IEEE, 2014, pp. 3088–3093.
[11] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, Y. Chen, OSA: An optical switching architecture for data center networks with unprecedented flexibility, Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association, 2012. 18–18.
[12] T. Wang, Z. Su, Y. Xia, M. Hamdi, Clot: A cost-effective low-latency overlaid torus-

based network architecture for data centers, Communications (ICC), 2015 IEEE International Conference on, IEEE, 2015, pp. 5479–5484.
[13] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, ACM SIGCOMM Computer Communication Review, 38 ACM, 2008, pp. 63–74.
[14] T. Wang, Y. Xia, J. Muppala, M. Hamdi, S. Foufou, A general framework for performance guaranteed green data center networking, Global Communications Conference (GLOBECOM), 2014 IEEE, IEEE, 2014, pp. 2510–2515.
[15] T. Wang, B. Qin, Z. Su, Y. Xia, M. Hamdi, S. Foufou, R. Hamila, Towards bandwidth guaranteed energy efficient data center networking, Journal of Cloud Computing 4 (1) (2015) 9, https://doi.org/10.1186/s13677-015-0035-7.
[16] N.R. Adiga, M.A. Blumrich, D. Chen, P. Coteus, A. Gara, M.E. Giampapa, P. Heidelberger, S. Singh, B.D. Steinmacher-Burow, T. Takken, et al., Blue gene/l torus interconnection network, IBM J. Res. Dev. 49 (2.3) (2005) 265–276.
[17] R. Alverson, D. Roweth, L. Kaplan, The gemini system interconnect, High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on, IEEE, 2010, pp. 83–87.
[18] D. Chen, N.A. Eisley, P. Heidelberger, R.M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, J. Parker, The ibm blue gene/q interconnection fabric, Micro, IEEE 32 (1) (2012) 32–43.
[19] W.J. Dally, H. Aoki, Deadlock-free adaptive routing in multicomputer networks using virtual channels, Parallel and Distributed Systems, IEEE Transactions on 4 (4) (1993) 466–475.
[20] L.G. Valiant, G.J. Brebner, Universal schemes for parallel communication, Proceedings of the thirteenth annual ACM symposium on Theory of computing, ACM, 1981, pp. 263–277.
[21] L. Gravano, G.D. Pifarre, P.E. Berman, J.L. Sanz, Adaptive deadlock-and livelock-free routing with all minimal paths in torus networks, Parallel and Distributed Systems, IEEE Transactions on 5 (12) (1994) 1233–1251.
[22] N. Farrington, E. Rubow, A. Vahdat, Data Center Switch Architecture in the Age of Merchant Silicon, IEEE Hot Interconnects, (2009). New York
[23] W. Dally, B. Towles, Principles and practices of interconnection networks, Morgan Kaufmann, 2004.
[24] http://www.amazon.com/.
[25] http://www.nsnam.org.