Random Links in Network Topology

Koibuchi12 - A Case for Random Shortcut Topologies for HPC Interconnects

Hassan Mango

Main idea

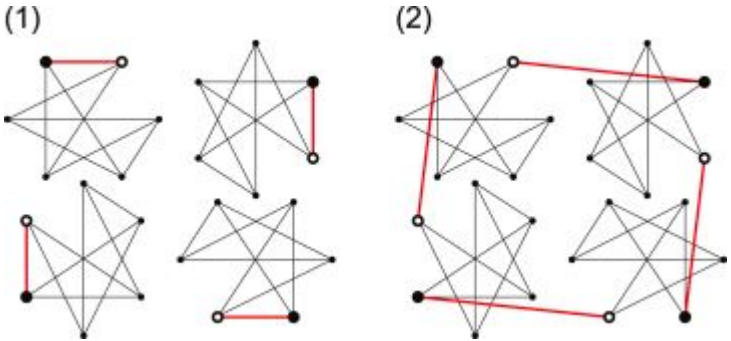
As the scales of parallel applications and platforms increase, the negative impact of communication latencies on performance, becomes large.

Which means, HPC systems are supposed to withstand applications without having to deal with problems like bottlenecks, deadlocks and improper load balancing.

Fortunately, modern High Performance Computing (HPC) systems can exploit low-latency topologies of high-radix switches.

In this context, the Koibuchi research paper proposes the use of random shortcut topologies generated by augmenting the usual topologies with random links between the groups.

Graph optimization algorithm for low-latency interconnection networks



However, what if we try to minimize the random links?

Topologies Used

→DLN

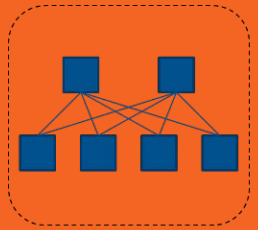
→Torus

→Flattened Butterfly

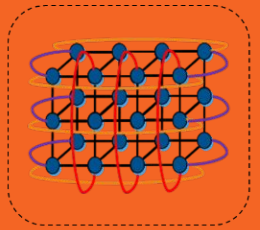
→HyperCube

→Mesh

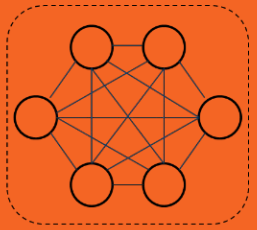
Basically, we are adding random links to prebuilt layers of network topologies to see how it affects the diameter and number of nodes



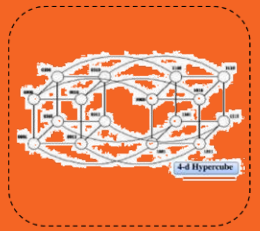
Fat Tree



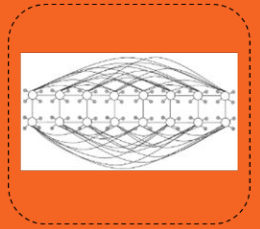
Torus



Dragonfly



Hypercube



HyperX

How would adding random links affect the topology?

Adding random links can significantly reduce the diameter of the topology.

But...

with random links, the communication patterns between nodes and sections can be unpredictable.

Previous

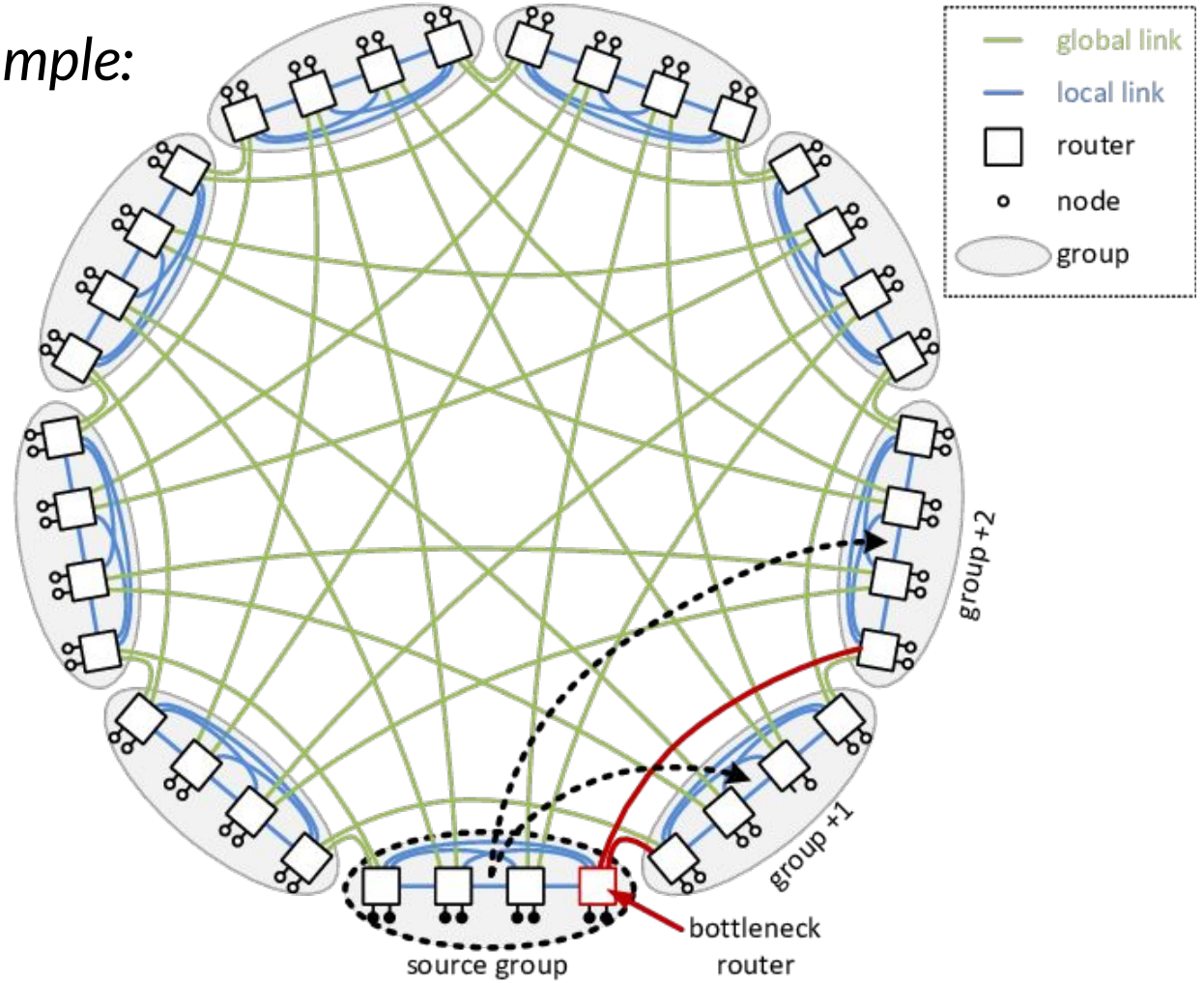
ideas for non-random graphs

1. Nodes in a ring topology are divided into sections, and each section has shortcuts connecting nodes to other sections. Nodes in each section connect to nodes in adjacent sections, creating efficient communication paths.

2. Sections of nodes have shortcuts to nodes in other sections. Nodes in each section can dynamically switch between connecting to the nearest or furthest available section, optimizing network performance.

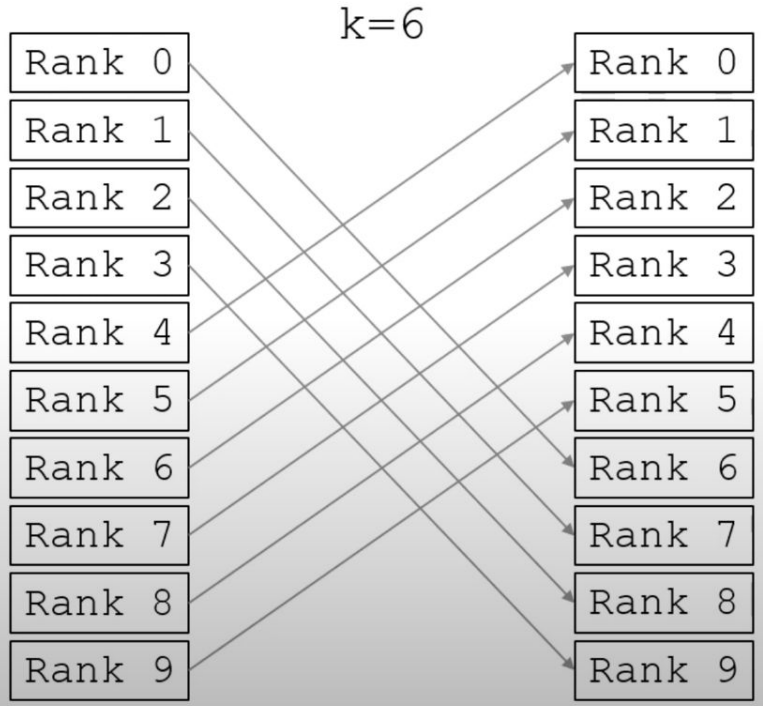
3. Sections are divided into left, middle, and right subsections. Each subsection has evenly distributed shortcuts to nodes in other sections, ensuring balanced and efficient communication within and between subsections.

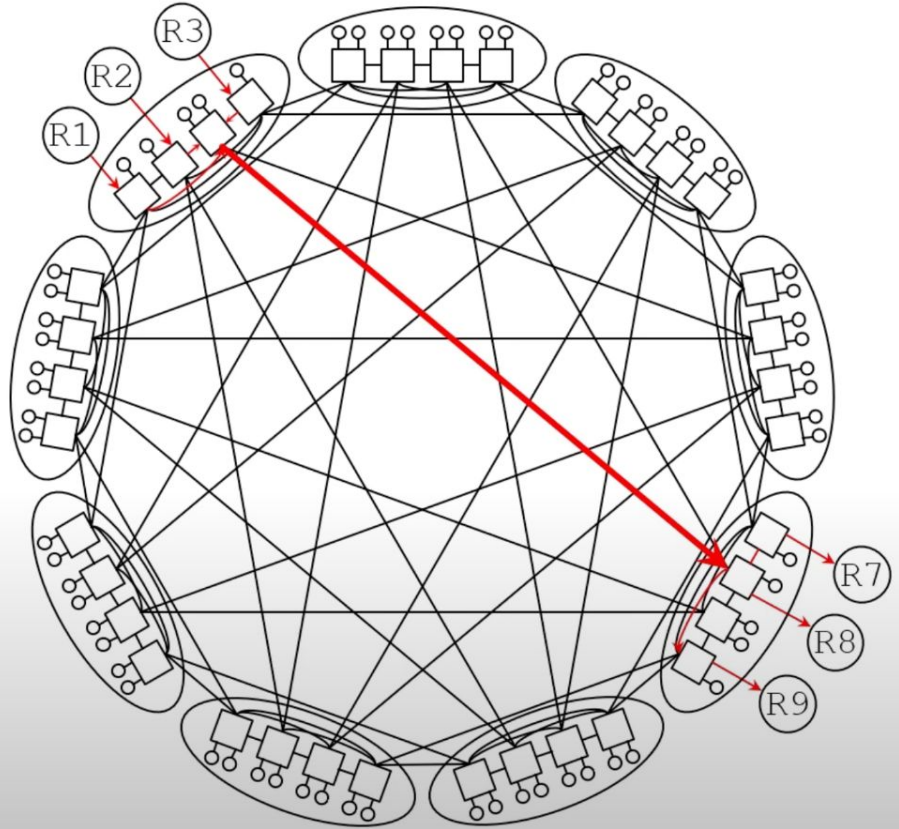
Bottleneck Example:



Bottleneck: Intra-Job Interference

- **Pairwise Algorithm**
  Process$[i]$ sends data to process$[i + k]$,

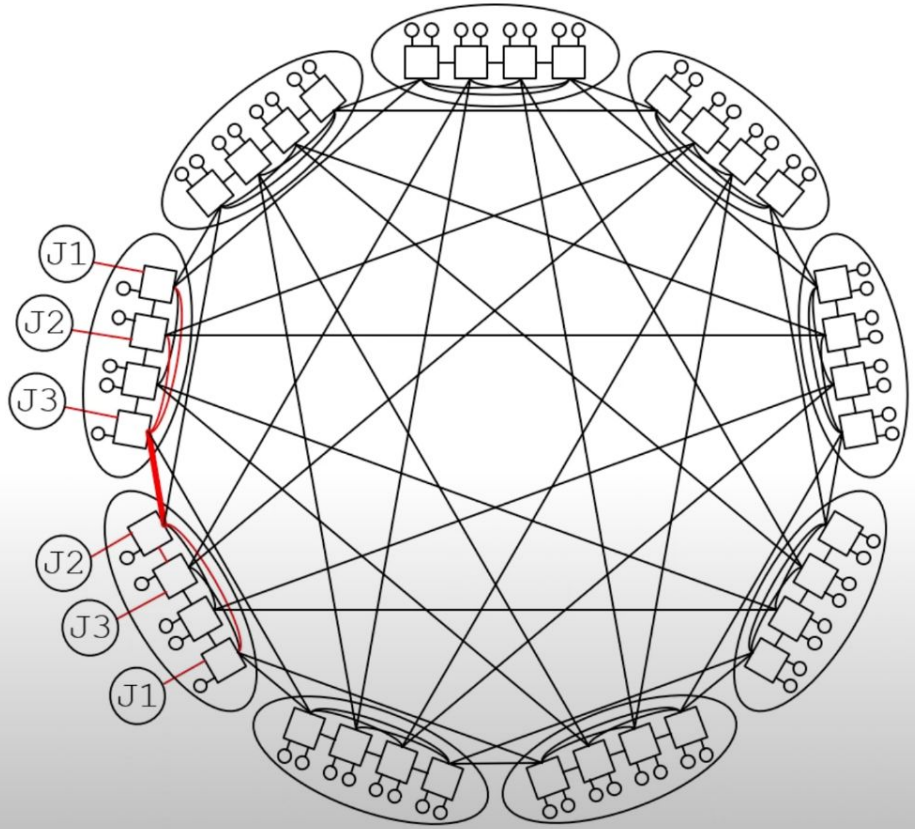




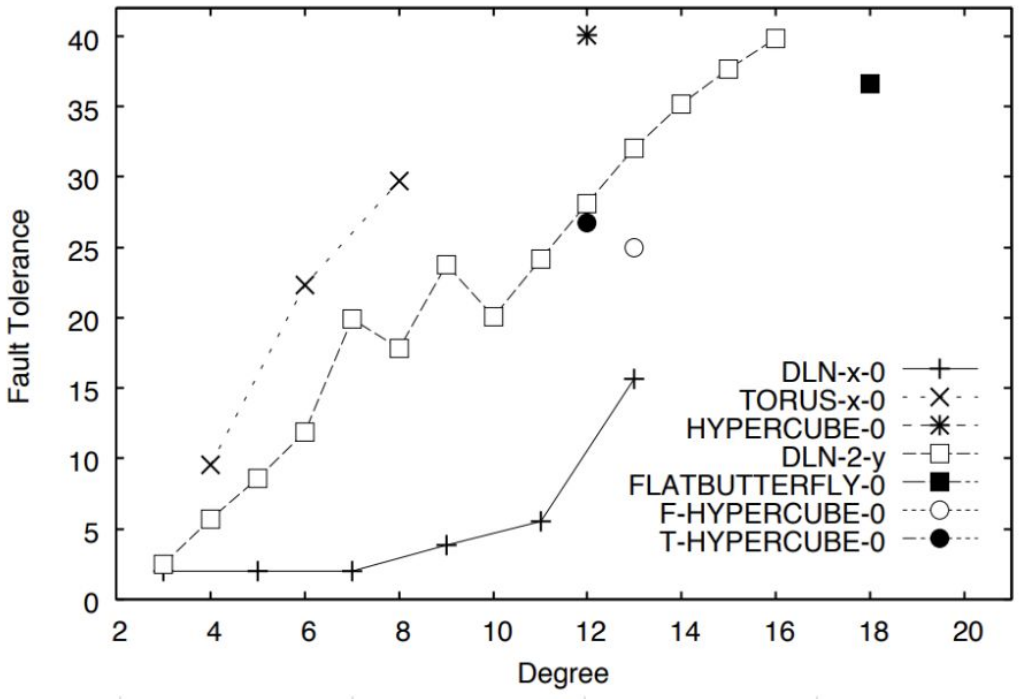
Bottleneck: Inter-Job Interference

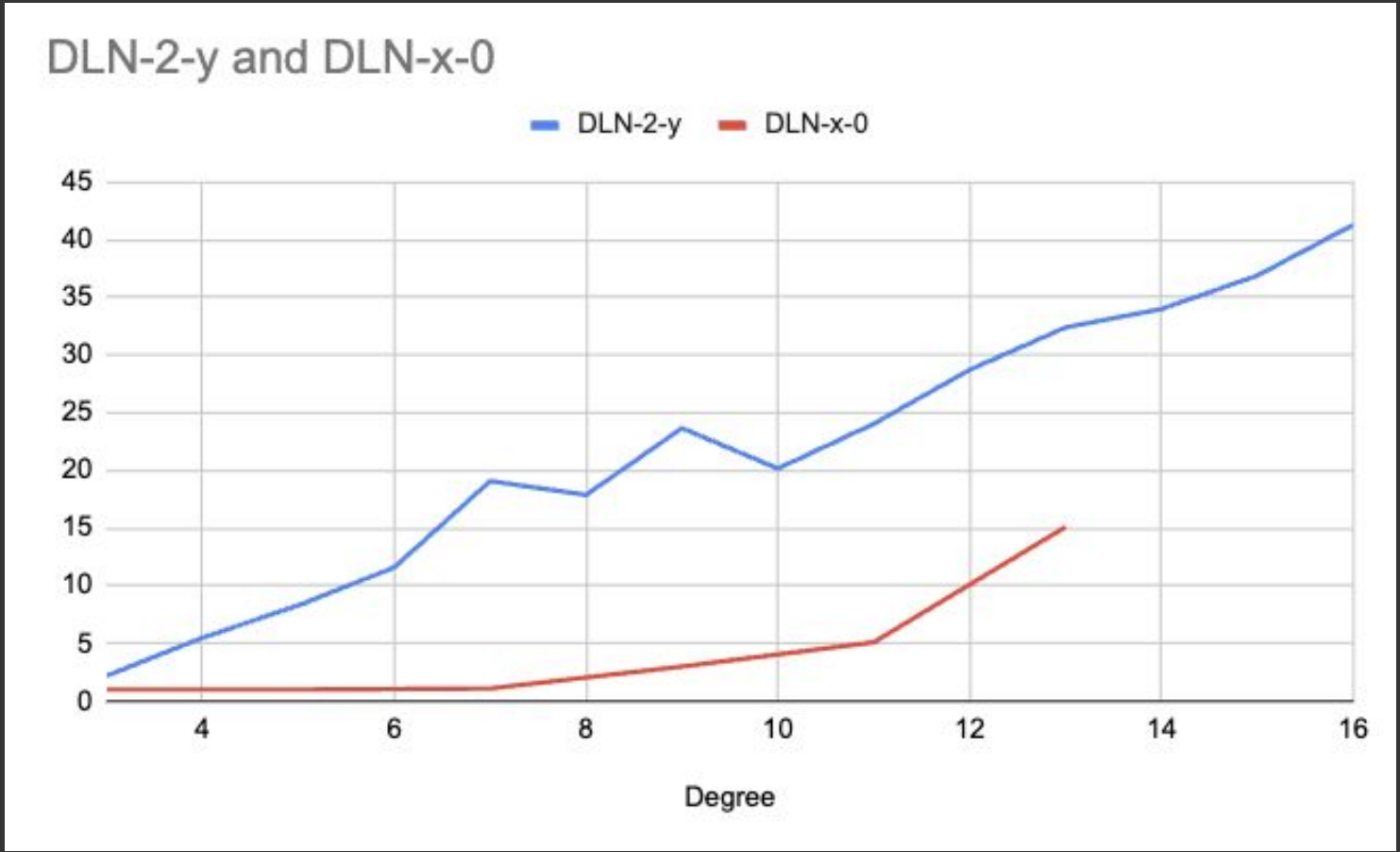
- Ring Algorithm
- Neighbor Exchange Algorithm

Only communicate with neighbor groups.



This figure shows the relationship between the degree and fault tolerance of each topology.

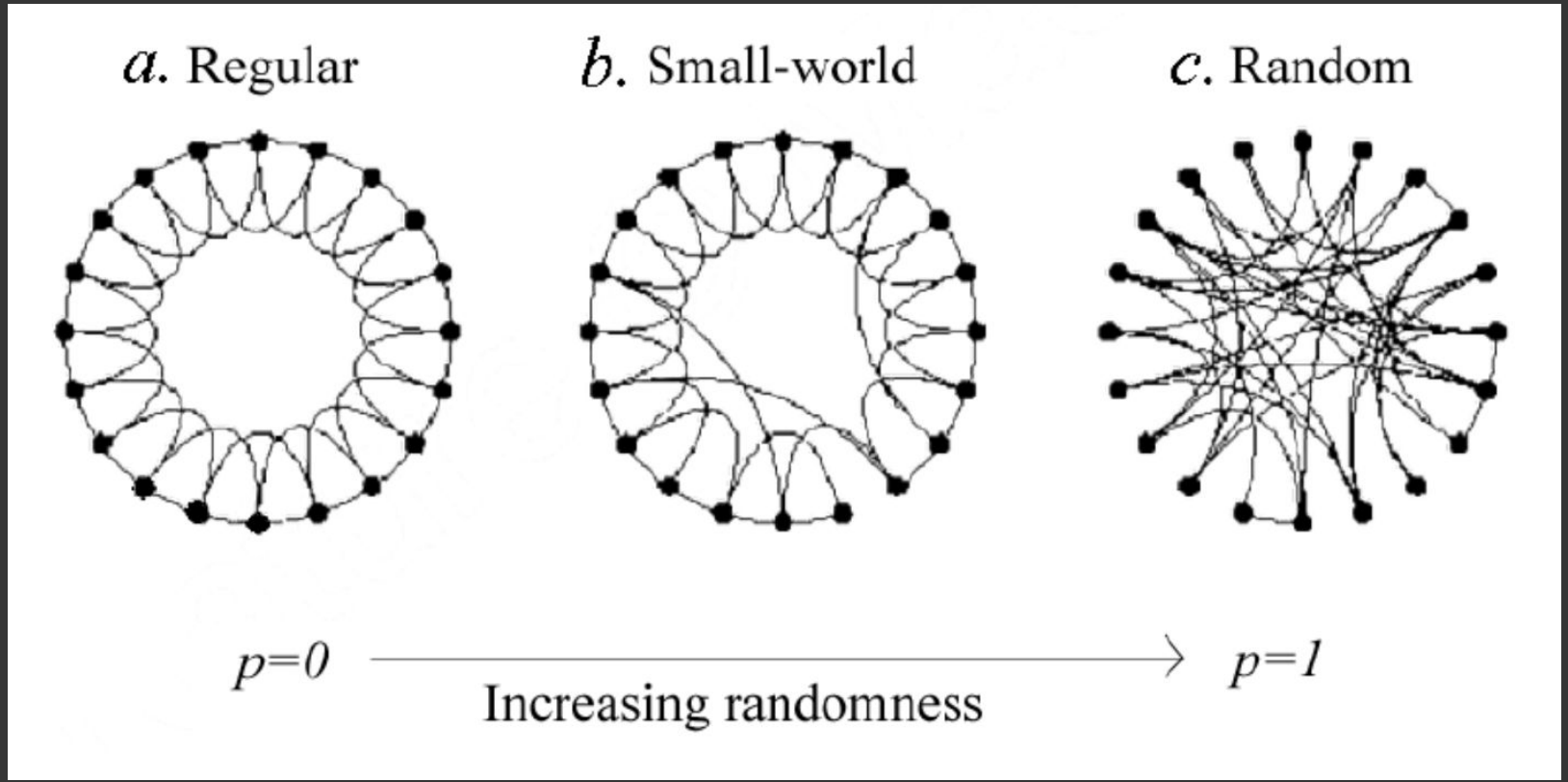



DLN-2-y and DLN-x-0
DLN-2-y
DLN-x-0
45
40
35
30
25
20
15
10
5
0
4
6
8
10
12
14
16
Degree

At greater values of N

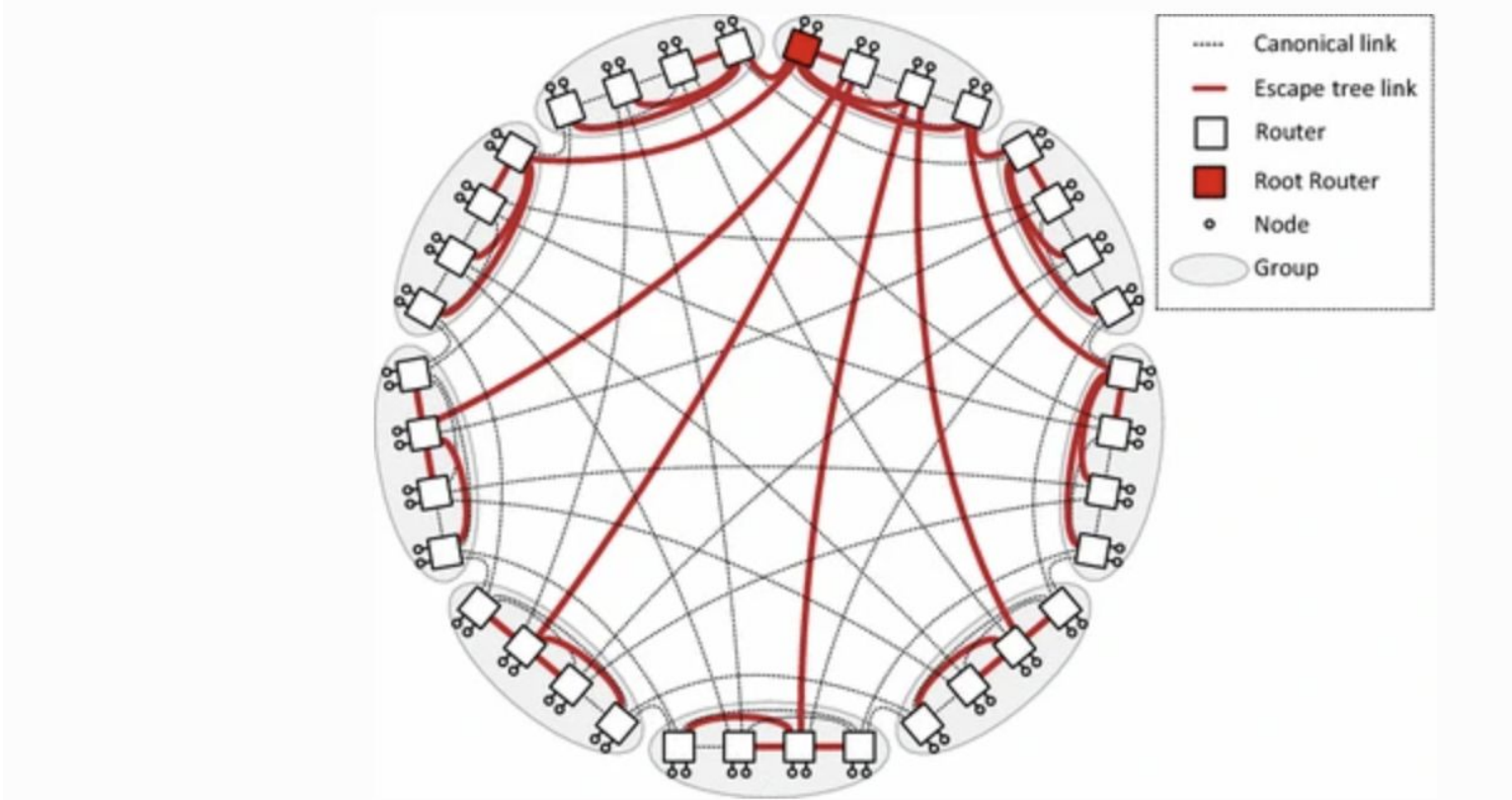
$\approx N$	T3D	T5D	HC	LH-HC	FT-3	DF	FBF-3	DLN	**SF**
512	30%	-	40%	55%	35%	-	55%	60%	**60%**
1024	25%	40%	40%	55%	40%	50%	60%	-	**-**
2048	20%	-	40%	55%	40%	55%	65%	65%	**65%**
4096	15%	-	45%	55%	55%	60%	70%	70%	**70%**
8192	10%	35%	45%	55%	60%	65%	-	75%	**75%**

At the same value of N



Suggestions?

+ ‘Roundabout station’
+ Deadlock free escape (path that tasks take when all nodes are occupied), that interconnects all nodes in the network.
+ Single vs multiple escape paths.



Dragonfly network $h = 2$ with an additional escape tree subnetwork

Real life example:Devices connected to Routers on campus

Can devices and routers on our network be connected the same way in the paper?

Any other example?