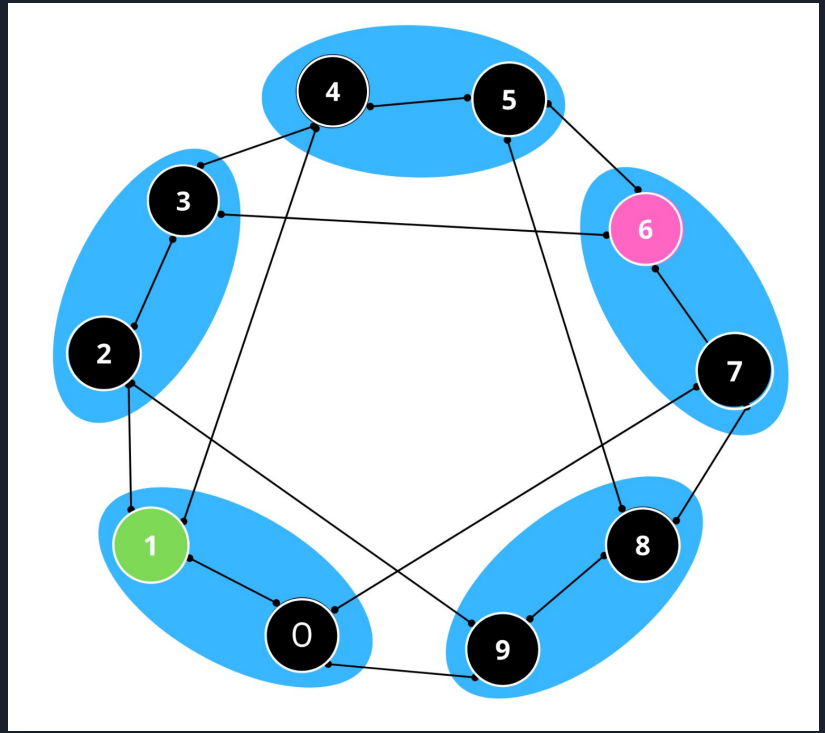
Summer Research: DragonFly Topology

Ridham Dholaria, Pedro Lopez

John Kim, William J. Dally, Steve Scott, Dennis Abts, Technology-Driven, Highly-Scalable Dragonfly Topology*

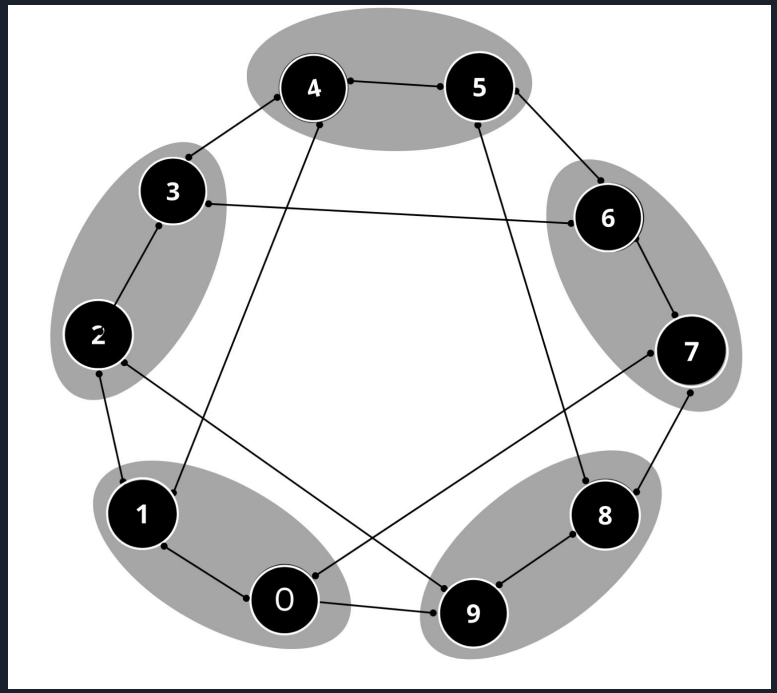
What is DragonFly?

- Max distance 3
- (processors per group, global connections per processor)
- Different arrangements possible



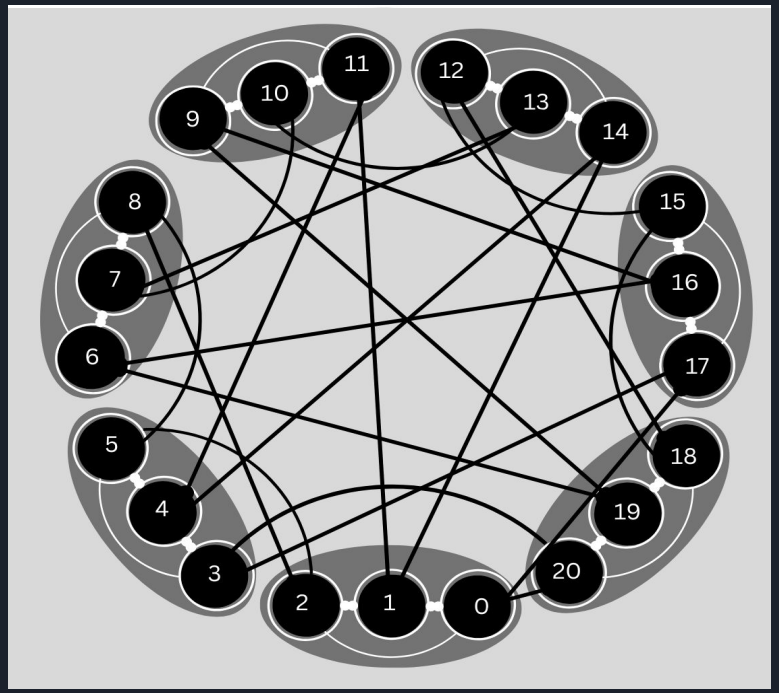
Absolute Global Link Arrangement (2,2)

DragonFly Example



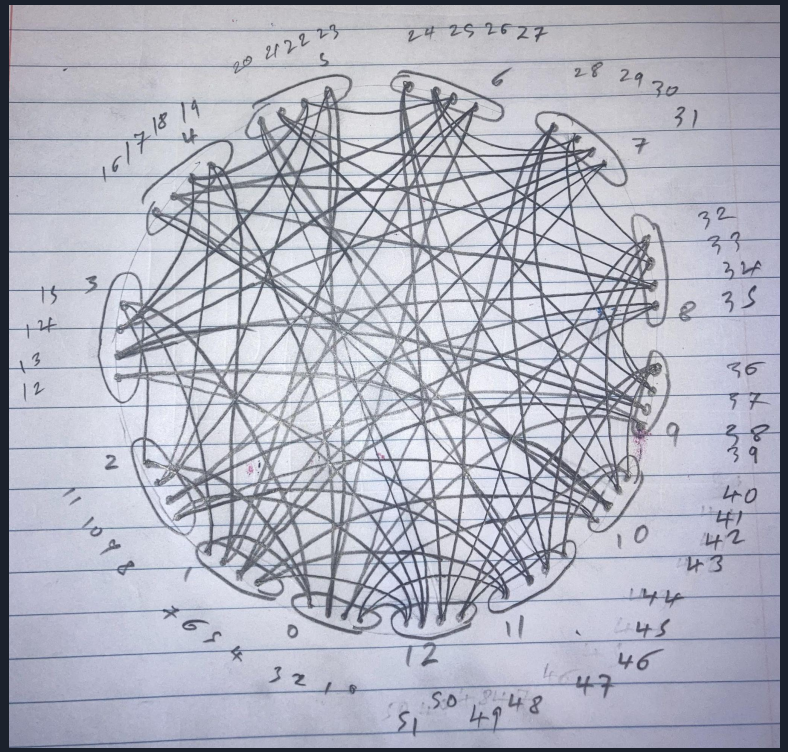
Absolute Global Link Arrangement (2,2)

DragonFly Example

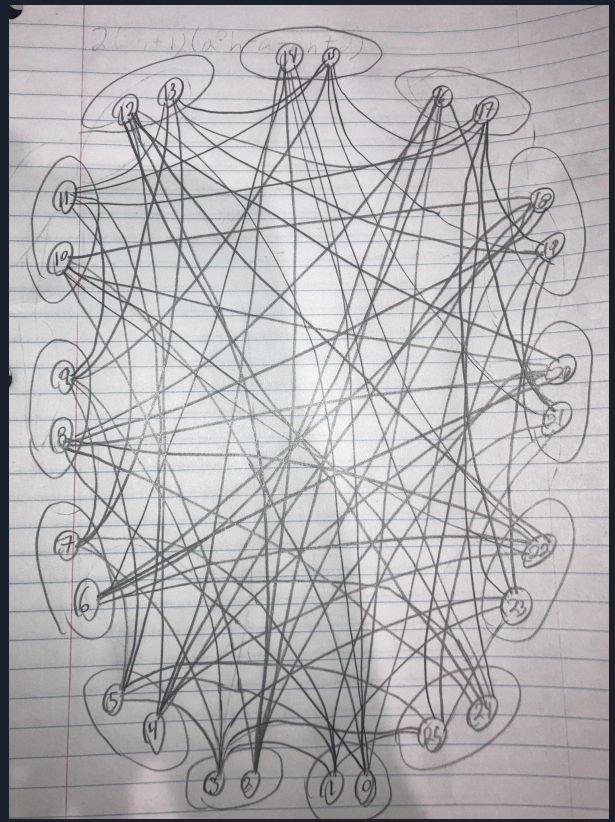


Absolute Global Link Arrangement (3,2)

Complex Topologies we hand drew



Absolute (4,3)



Circulant (2, 6)

Different DragonFly Topologies Implementation

Emily Hastings, David Rincon-Cruz, Marc Spehlmann, Sofia Meyers, Anda Xu, David P. Bunde. Comparing global link arrangements for Dragonfly networks

Madison Belka, Myra Doubet, Sofia Meyers, Rosemary Momoh, David Rincon-Cruz, David P. Bunde. New link arrangements for Dragonfly networks

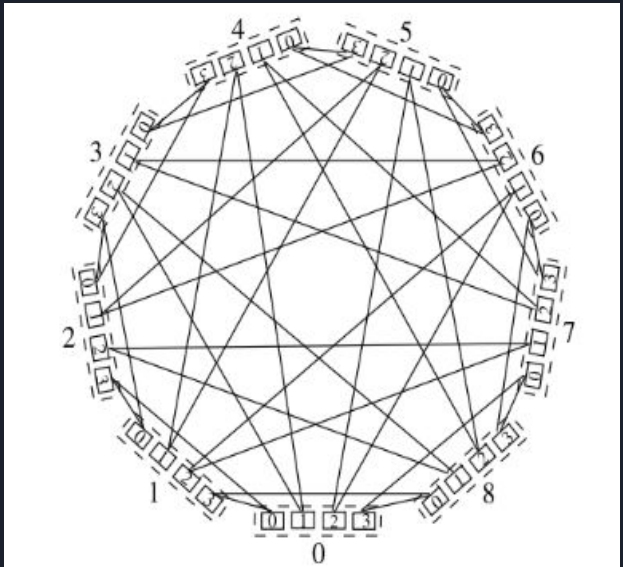
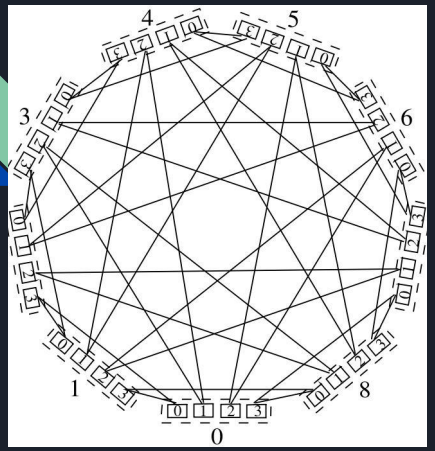
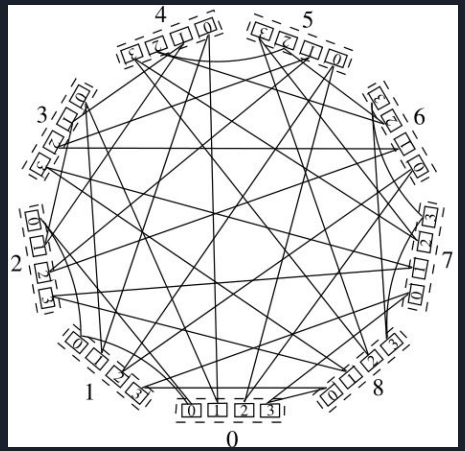


Figure 3. Relative global link arrangement for $(p, 4, 2)$

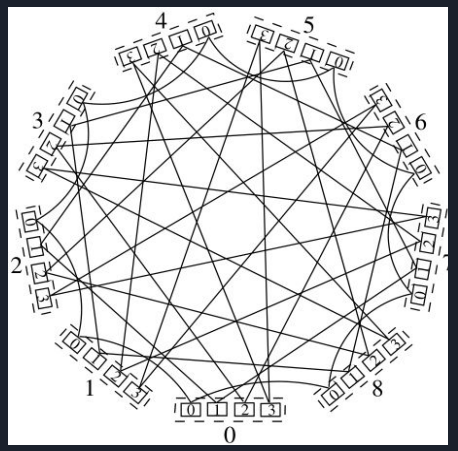
- Different ways of connecting DragonFly graphs:
  - Absolute
  - Relative
  - Circulant
  - Helix
  - Nautilus



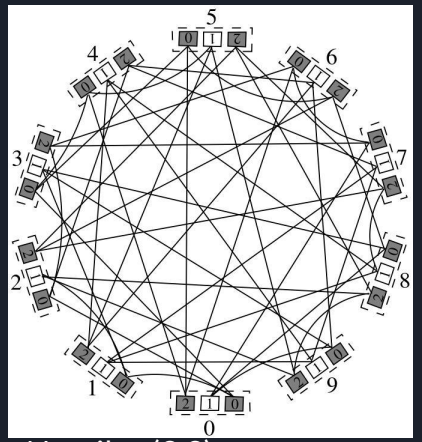
Absolute (4,2)



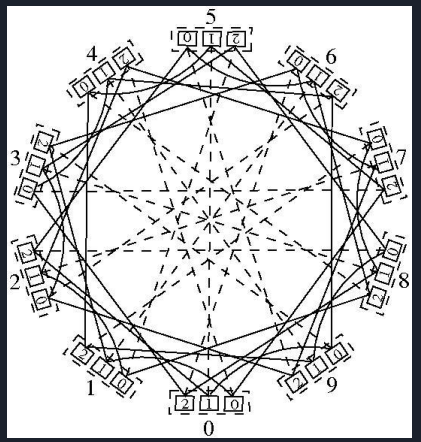
Relative (4,2)



Circulant (4,2)



Nautilus (3,3)



Helix (3,3)

How are Arrangements different?

- **Absolute Arrangement**: Connects port K of group i -> group K if K < i and to group K + 1 otherwise
- **Relative Arrangement**: Port K of group i -> group (i + K + 1)
- **Circulant Arrangement: “h must be even.”** Port K of group i -> to group (i+K/2+1) if K is even and group (i−⌊K/2⌋−1) if K is odd

Important terms:
a - processors per group
h - global connections per processor
K - outgoing ports
i - current group

Part of our code which simulates Relative Arrangement

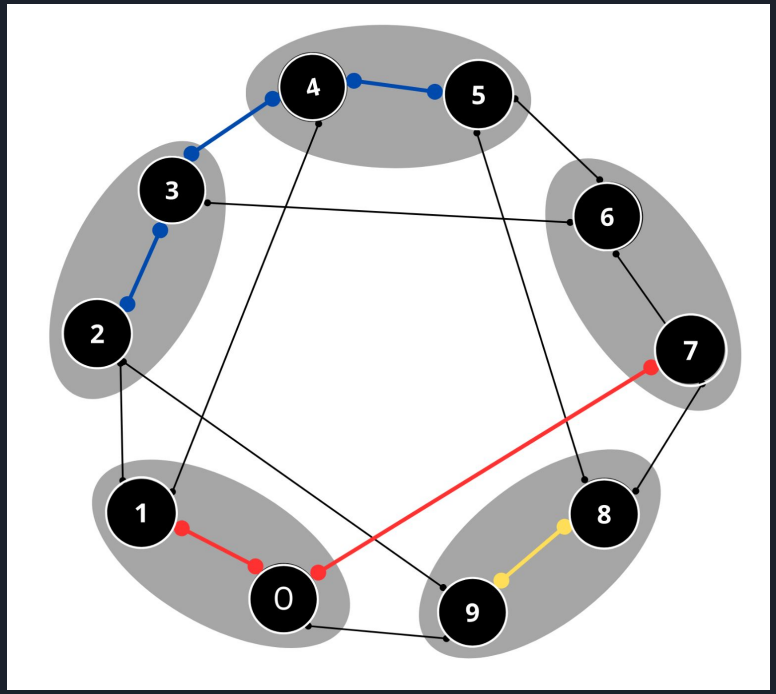
```
public void relativeArrange(int a, int h) {
  int g = (a * h) + 1, currSwitch = 0;
  Long time1 = System.currentTimeMillis();
  while (!(currSwitch > ((g * a) - 1))) {
    // System.out.println("relative("+a+","+h+") currSwitch: " + currSwitch);
    int[] portArray = new int[h];
    int[] groupArray = new int[h];
    int[] iPortArray = new int[h];

    //Step 1: Calculating Outgoing group
    int currGroup = currSwitch / a;

    //Step 2: Calculating Outgoing port
    int ports = (a * h) - 1;
    for (int i = (currGroup * a); i < ((currGroup * a) + a); i++) {
      for (int j = 0; j < h; j++) {
        if (i == currSwitch) {
          portArray[j] = ports;
          if (j == (h - 1)) {
            break;
          }
        }
        ports--;
      }
    }
```

```
public void relativeArrange(int a, int h) {
    }

    //Step 3: Calculating Incoming group
    for (int i = 0; i < h; i++) {
      int temp1 = currGroup + portArray[i] + 1;
      if (temp1 > (a * h)) {
        temp1 = temp1 - (a * h) - 1;
      }
      groupArray[i] = temp1;
    }

    // System.out.println("currSwitch "+currSwitch);
    //Step 4: Calculating Incoming ports
    int totalPorts = (a * h) - 1;
    for (int i = 0; i < h; i++) {
      iPortArray[i] = totalPorts - portArray[i];
      // System.out.println("iPortArray : " + iPortArray[i]);
    }
    //    System.out.println();
```

```
    //Step 5: Calculating Incoming switch
    // outerloop:
    for (int i = 0; i < h; i++) {
      // System.out.println("currSwitch: "+ currSwitch + " outerloop h: " + h);
      int temp = ((groupArray[i]) * a);
      ports = (a * h) - 1;
      outerloop:for (int j = temp; j < (temp + a); j++) {
        for (int t = 0; t < h; t++) {
          if (
            ports == iPortArray[i] && (!(checkNodeConnections(j, currSwitch)))
          ) {
            addBiEdge(currSwitch, j);
            break outerloop;
          }
          ports--;
        }
      }
    }
    currSwitch++;
  } //While loop ending
```

Distances in Graph

- Distance 0
- Distance 1 -> Yellow
- Distance 2 -> Red
- Distance 3 -> Blue



Relative Global Link Arrangement (2,2)

Total Distance calculation:

g = Total Groups in Graph: (a * h) + 1

Distance 0: Nodes Reachable in 0 hops
= 1 for every Node

Total Distance 0 Occurrences in a graph:
= Total # of Nodes * Dist 0 Nodes
= a(g) * 1

Distance 1: Nodes Reachable in 1 hop
= Local connections + Global connections
= (a - 1) + h

Total Distance 1 Occurrences in a graph:
= Total # of Nodes * Dist 1 Nodes
= a(g) * (a - 1) + h

Distance 2: Nodes Reachable in 2 hops
= (Local connections + Global connections) * (Local connections + Global connections)
= (a - 1) + h) * (a - 1) + h)

Total Distance 2 Occurrences in a graph:
= Total # of Nodes * Dist 2 Nodes
= a(g) * (a - 1) + h) * (a - 1) + h)

Ideally

Absolute Arrangement (4,2)		Relative Arrangement (4,2)		Circulant Arrangement (4,2)	
Distances	Occurrences	Distances	Occurrences	Distances	Occurrences
0	36	0	36	0	36
1	180	1	180	1	180
2	**900**	2	**900**	2	**900**

Distance 2 Discrepancy

Absolute Arrangement (4,2)		Relative Arrangement (4,2)		Circulant Arrangement (4,2)	
Distances	Occurrences	Distances	Occurrences	Distances	Occurrences
0	36	0	36	0	36
1	180	1	180	1	180
2	**456**	2	**486**	2	**468**

What is an overlap?

- Red - Distance 1
- Green - Distance 2
- Yellow - Overlapping Distance 2

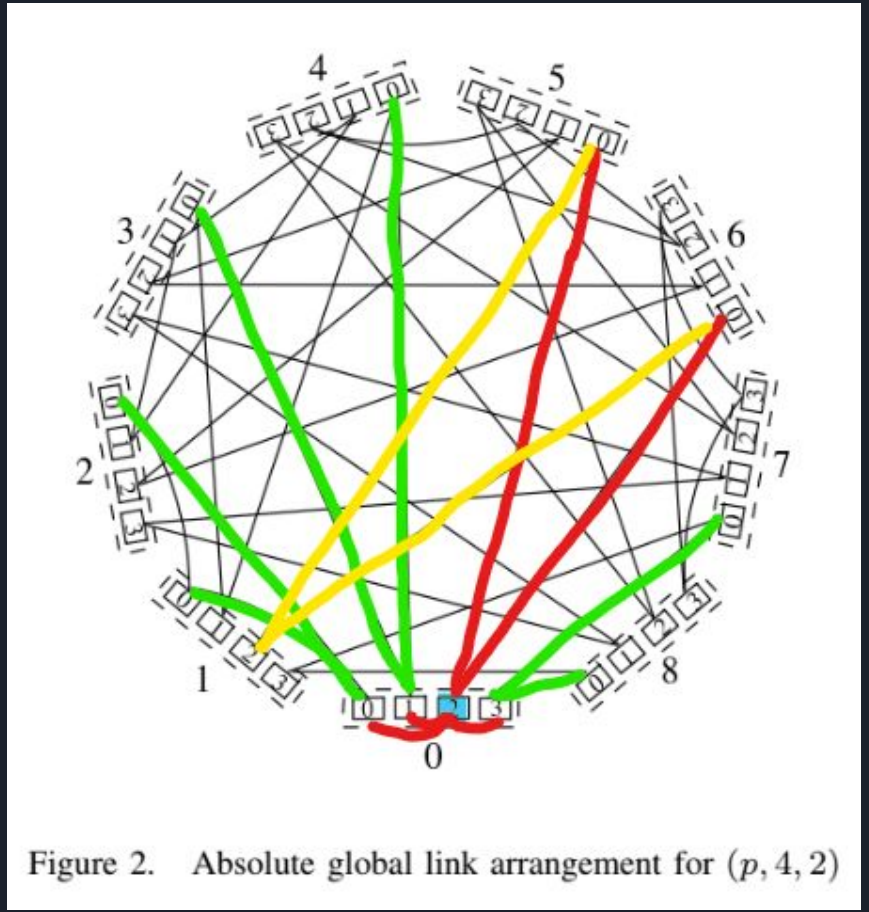


Figure 2. Absolute global link arrangement for $(p, 4, 2)$

Absolute Distance 2 - Formula

$$ah(a-1)\left(h^2+2ah-3h-(h-1)^2\right)+a(h+1)\left(h^2+2ah-3h-(h-1)^2-(h-1)\right)$$

Or

$$ah\left(2a^2h-ah+a-h-1\right)$$

(Simplified Formula)

Relative Distance 2 - Formula

$$2(ah+1)(2h(a-1))+((a-2)(ah+1))(2h(a-1)+2(h-1))$$

Or

(Simplified Formula)

$$2(ah+1)\left(a^2h-a-2h+2\right)$$

Future Work

- Circulant Equation
- Why Dist 2 is different
- Fault Tolerance

Thank You!