Interconnection Networks Summative

wcrr51

2021

Part 1

Question 1

1. Pre-building routes and storing them in routing tables at the nodes.

One of the primary overheads when storing the routing tables at nodes is that this uses storage space, meaning that, for any arbitrary (or naive) interconnection network, each node must store the route for (or at minimum the next node in the route) for any other node, meaning the table size is between linear and quadratic in the number of nodes. This leads to the primary difficulty when the interconnection network must be fault-tolerant in that if a node goes offline, all the other nodes (and their respective routing tables) will be ignorant of this change, meaning some routes may be blocked (and thus packets are lost). In this case, to remain fault-tolerant, one must use additional resource overhead to detect faults and correspondingly update the routing tables for all nodes (preventing program-specific messages being sent). Other difficulties include that tables may be large which incur a lookup overhead, using valuable time that could be spent on processing, and that each node must have a unique lookup table which may take time to implement / update.

2. Building routes on-the-fly.

If the neighbours of a faulty node are aware of the issue, they can dynamically adjust their route to avoid the node, and thus avoid lost packets. However, building routes on-the-fly carries additional processing overhead, as a message arrives at or is generated by each node, some processing power (and time) needs to be spent working out which node it should be sent to next. While this processing is taking place, the message is stationary meaning higher latency (and less task-specific processing is being done). The main difficulty with building routes on-the-fly is that the interconnection network needs to be well-described (as opposed to arbitrary/random), which incurs extra design complexity (especially when incorporating path diversity for fault-tolerance). This has the added difficulty in that simplifying the algorithms to make them faster makes them less versatile (potentially sacrificing desirable properties such as load balancing). Furthermore, simplistic on-the-fly algorithms may lead to livelock.

Question 2

(a)

Using the fact that the network is symmetric, the table will be the same size for all the nodes.

1. There are 4 nodes of minimal distance 1 from the source, requiring $4 \cdot 4 \cdot 1 = 16$ bits.
2. There are 6 nodes of minimal distance 2 from the source, requiring $6 \cdot 4 \cdot 2 = 48$ bits.
3. There are 4 nodes of minimal distance 3 from the source, requiring $4 \cdot 4 \cdot 3 = 48$ bits.
4. There is 1 node of minimal distance 4 from the source, requiring $1 \cdot 4 \cdot 4 = 16$ bits.

Summing this up, there are 15 nodes (excluding the source) requiring 128 bits to be stored.

Interestingly, as an aside, the recursive nature of the n-cube means that the number of nodes of minimal distance i from a node is $\binom{n}{i}$. Hence, the above expression can be calculated with $4\sum_{i=0}^{n} i\binom{n}{i}$ which, for $n = 4, 4\sum_{i=0}^{4} i\binom{4}{i} = 4 \cdot (0 + 4 + 16 + 16 + 4) = 4 \cdot 32 = 128$ bits.

(b)

In a Q_4 there exist $2^4 = 16$ nodes, thus 16 table entries. As each table entry represents one node, 4 bits are required to store the target, and 4 bits are required to store the next node.

Hence, for a table consisting of the next path required to get to all nodes, $16 \cdot 8 = 128$ bits (or 64 bits if not storing the target node). As each node need not store a table entry for itself, and assuming neighbouring nodes may be routed to through other neighbours (i.e. the fastest path to a neighbour is not necessarily a direct hop to the neighbour), $128 - 8 = 120$ bits are required (or 60 bits if not storing the target node).

(c)

Each node is labelled by 4 bits, however this is only necessary to uniquely identify each node in a global context. Since the table from (b) only stores the next neighbour, and as each node has four neighbours, only two bits are required to locally identify them ($\{(0,0),(0,1),(1,0),(1,1)\}$). The easiest way to assign these exploits the algebraic definition of the hypercube node labels, and works as follows. As an edge exists between a node who's label differs in one bit (i.e. has Hamming distance 1), store the (index) position of that differing bit in the table as the binary number between 1 and 4.

For example, node $(0,0,0,0)$ has outbound and inbound channels to and from nodes $(1,0,0,0)$, $(0,1,0,0)$, $(0,0,1,0)$, and $(0,0,0,1)$, under this coding scheme, the table would store the nodes as $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$ respectively.

Using this coding scheme, the table only has to store a mapping from the 4-bit destination node to the 2-bit neighbouring node, meaning $16 \cdot 6 = 96$ bits for all nodes, including the sender (or 32 bits if not storing the target node). Hence, $96 - 6 = 90$ bits excluding the sender (or 30 bits if not storing the target node).

Question 3

(a)

The circulant graph $C(V_C, E_C), |V_C| = n, n \bmod 2 = 0, n \bmod 4 \neq 0$ is defined such that

$$V_C = \{0, 1, 2, ..., n-1\}$$

$$E_C = \{(u, v) \in V_C^2 : v = u + 1 \vee v = u + \frac{n}{2}\}$$

The Petersen graph $P(V_P, E_P), |V_P| = n, n \bmod 2 = 0, n \bmod 4 \neq 0$ is defined such that

$$V_C = \{0, 1, 2, ..., n-1\}$$

$$E_P = \{(u, v) \in V_P^2 : (u \bmod 2 = 0 \wedge (v = u + 1 \vee v = u + 2 \bmod n)) \vee (u \bmod 2 = 1 \wedge v = (u + 4) \bmod n)\}$$

(b)

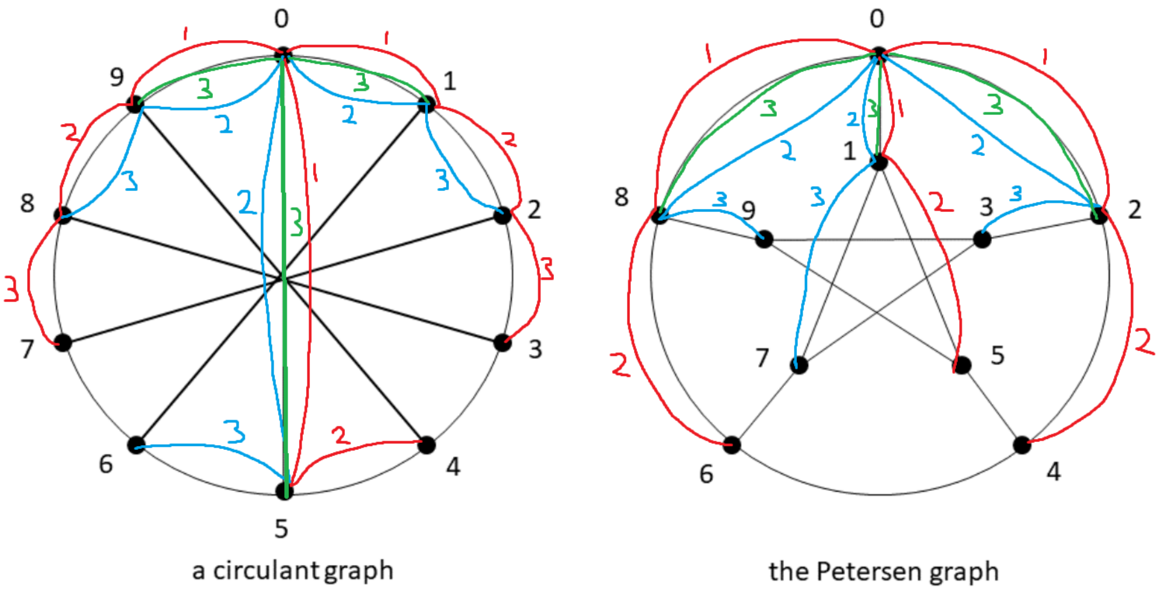

Figure 1: Broadcast schedules showing potential single-node scatter routings for a circulant graph and the Petersen graph.

The route node on each graph may only send out one message per neighbour at each time step (since each message is b bits and each outward channel can only handle b bits per second).

For the circulant graph, this means sending out messages to the furthest nodes from node 0 in the first second (these being 3 and 7, and, in this example, 4, though 6 works as well).

Once these have been dispatched to nodes 1, 9, and 5 respectively, in the second second, node 0 emits the next three messages bound for nodes 2, 8, and 6 to nodes 1, 9, and 5 respectively. Meanwhile nodes 1, 9, and 5 forward the original messages to nodes 2, 8, and 4 respectively. Node 4 now has its message.

In the third and final second, node 0 emits the messages belonging to its neighbours (1, 5, and 2). Meanwhile, nodes 2, 8, 5, 1, and 9 forward the messages on to nodes 3, 7, 6, 2, and 8 respectively. All nodes now have their messages after three seconds.

For the Petersen graph, in the first second, node 0 first dispatches the messages bound for nodes 4, 5, and 6 along the channels to nodes 2, 1, and 8 respectively.

In the second second, node 0 emits the messages bound for nodes 3, 7, and 9 along the channels for nodes 2, 8, and 1 respectively. Meanwhile, nodes 2, 1, and 8 forward the messages on to nodes 4, 5, and 6, respectively. Nodes 4, 5 and 6 now have their messages.

In the third second, node 0 emits the messages bound for its neighbours (nodes 2, 1, and 8) along the adjacent channels. Meanwhile, nodes 2, 1, and 8 forward the messages on to nodes 3, 7, and 9 respectively. All nodes now have their messages after three seconds.

These are both demonstrated visually in Figure 1, with the red path representing the first messages sent, the blue representing the second, and the green representing the third.

(c)

In the circulant graph, after the first second, the neighbours of node 0, nodes 1, 5, and 9, have received the message. After the second second, node 1 send the message to nodes 2 and 6, and node 9 send the message to nodes 4 and 8 (node 5 may also send the message to nodes 4 and 6). This means that at the start of the third second, all but nodes 3 and 7 have not received the message. Hence, a single-node broadcast is not possible in 2 seconds on the circulant graph of $n = 10$. As the graph is of diameter 3, the breadth-first search tree is of height 3, meaning not all nodes can be reached within 2 seconds.

In the Petersen graph, after the first second, the neighbours of node 0, nodes 1, 2, and 8, have received the message. After the second second, node 1 sends the message to nodes 5 and 7, node 2 sends the message to nodes 3 and 4, and node 8 sends the message to nodes 6 and 9. This means that at the start of the third second, all nodes have received the message. As the graph is of diameter 2, the breadth-first search tree being of height 2 enables all nodes to be reached within 2 seconds. This is demonstrated without labelled timings in Figure 2.

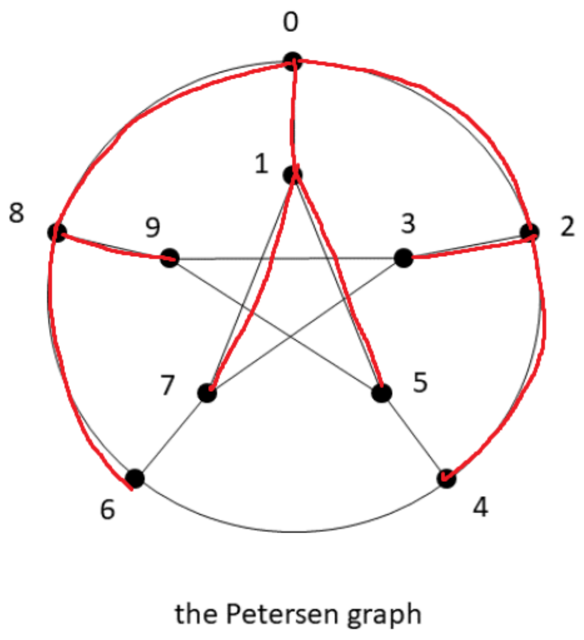

Figure 2: Routing for the Petersen graph.

(d)

A trivial routing for the circulant graph is to simply pass messages around the circle until everyone has received theirs. However this leads to high and unbalanced loads (since the $(v, (v + 5) \bmod 10)$ inbound and outbound channels are not being used.

The single-node scatter shown in Figure 1 can be used as a routing for all nodes in a total exchange.

We only need to consider a single circular $((v, (v+1) \bmod 10))$ channel in one direction due to the trivial rotational and reflective automorphisms. Hence, in accordance with Figure 1, we consider the channel from node 0 to 1. In the first second, only the message from node 0 to 1, bound for node 3, is sent. In the second second, the next message from node 0 to 1, bound for node 2, is sent, simultaneously node 0 forwards the message from node 9 to node 1 (bound for node 2). In the third second, the message from node 0 to 1 is sent, node 0 also forwards the messages from nodes 5, 8, and 9 bound for node 1. This means that the channel from node 0 to 1 has 4 message paths crossing though it. Hence, all channels $(v, (v+1) \bmod 10)$ and $(v, (v-1) \bmod 10)$ have a load of 4.

Now consider the channels crossing the circle, these only have one message being sent down them at any time (directly from the source node), meaning all channels $(v, (v+5) \bmod 10)$ and $(v, (v-5) \bmod 10)$ have a load of. While this is more balanced than the trivial routing, it is still not great! To improve, one could send messages bound for nodes 4 and 6 from 0 through nodes 9 and 1 respectively (which would take more time but lead to loads of 4 for circular channels and 3 for cross-circle channels).

For the Petersen graph, as it is node- and channel-symmetric, reasoning about the loads on one channel also apply to all other channels. This means that the load will be the sum of all the channels being used by a single-node scatter routing. Assuming the single-node scatter routing is done as it were in the single-node broadcast shown in Figure 2, this would amount to a load of 9 on every channel.

Part 2

Question 4

(a)

$B_4(V_B, E_B)$ is formed such that

$$V_B = \{(a,b,c,d) \in \{1,2,3,4\}^4 : a \notin \{b,c,d\}, b \notin \{a,c,d\}, c \notin \{a,b,d\}, d \notin \{a,b,c\}\}$$

$$\begin{aligned} E_B = \{((a,b,c,d),(x,y,z,w)) \in V_B^2 :& (a = y \wedge b = x \wedge c = z \wedge d = w) \vee \\ & (a = x \wedge b = z \wedge c = y \wedge d = w) \vee \\ & (a = x \wedge b = y \wedge c = w \wedge d = z)\} \end{aligned}$$

(b)

Consider how each edge in the graph represents a swap between two adjacent elements (numbers) in the vertex label. The hop count between maximum number of swaps must be the diameter of the graph as the maximum number of swaps represent the furthest nodes apart. As the maximum number of swaps occurs when a node label is reversed ((A,B,C,D) to (D,C,B,A)), as with bubble sort, at least six swaps must be made (three to move the left-most element to the right, two to move the new leftmost element to the second right-most position, and one to swap the first and second element). Hence, the diameter of B_4 is 6.

(c)

For B_4 to be a Moore graph, given its out-degree $\delta_0 = 3$, level $i \geq 0$ of its breadth-first search tree would need to have $\delta_0(\delta_0 - 1)^{i-1} = 3 \cdot 2^{i-1}$ nodes. As the graph is symmetric, the root node for the breadth-first search does not matter. Performing this breadth-first search from node 4321 firstly identifies the level $i = 1$ nodes $|\{4231, 3421, 4312\}| = 3$, which holds for $3 \cdot 2^{1-1} = 3 \cdot 2^0 = 3$. However, expanding the tree to level $i = 2$ identifies nodes $|\{4213, 2431, 3241, 3412, 4132\}| = 5$, and as $3 \cdot 2^{2-1} = 3 \cdot 2 = 6 \neq 5$, as the condition does not hold, B_4 is not a Moore graph.

(d)

The number of node-disjoint paths is upper-bounded by the graph's in and degree of 3 and out degree of 3 for all nodes, as seen by inspection. Hence, three viable node-disjoint paths, as shown visually in Figure 3, are as follows:

$$\begin{aligned} P_1 =& [4312, 3412, 3142, 3124, 3214, 2314] \\ P_2 =& [4312, 4132, 1432, 1342, 1324, 1234, 2134, 2314] \\ P_3 =& [4312, 4321, 3421, 3241, 2341, 2314] \end{aligned}$$

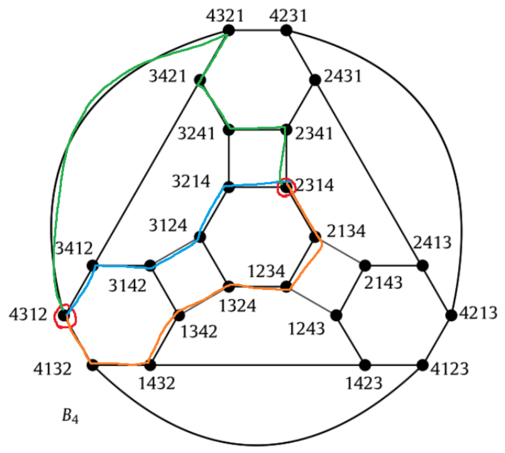

Figure 3: P_1 in blue, P_2 in orange, and P_3 in green node-disjoint paths between nodes 4312 and 2314 in B_4.

Geometrically, there is an reflective automorphism vertically down the middle of B_4 as is presented in the picture. Additionally, there is a 120 degree rotational automorphism about the geometric center of B_4 as is presented in the picture. Based on this, nodes 3214 and 4213, and 1234 and 4231 as shown in Figure 4 can be acquired from automorphisms on 4312 and 2314.

4312 and 2314 maps to 3214 and 4213 respectively under the reflection automorphism, and 4312 and 2314 maps to 1234 and 4231 respectively under the rotational automorphism.

This implies that, because 4312 to 2314 have three node-disjoint paths between them, the other two node pairs must each have three node-disjoint paths between them as well.

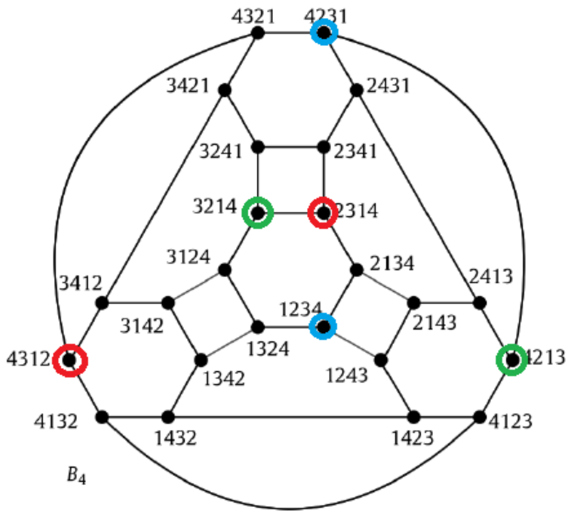

Figure 4: Nodes 4312 and 2314 (red), 3214 and 4213 (green), and 1234 and 4231 (blue) in B_4.

(e)

B_4 can be re-drawn as a truncated octahedron (see Figure 5 which is clearly an instance of order-4 Cayley graph under the symmetric group S_4, this can be seen in Figure 6 (B_4 is also isomorphic to the order-4 permutohedron). While the core symmetric properties of the S_4 group alone show that B_4 is node-symmetric, we consider some simple geometric automorphisms that can map one node to any other node so as to prove the node-symmetry of B_4.

The truncated octahedron has many reflective symmetries, these include reelections across the planes rotated 0, 120, and 240 degrees about the oppositely oriented hexagons, and across the planes rotated 0, 90, 180, and 270 degrees about the oppositely oriented hexagons.

However, one only needs to consider rotational symmetries to map one node to any other. It is clear to see that the truncated octahedron can be rotated about 3-dimensional lines intersecting the 3 pairs of oppositely oriented squares (90, 180, and 270 degrees), and 4 pairs of oppositely oriented hexagons (120 and 240 degrees). These five automorphisms alone can be combined into one that maps any node to any other node in up to 6 rotations, one example of this is as follows:

1. Rotate the given node to its correct position in one of its two adjacent hexagons.
2. Rotate it about to the correct position within its local square.
3. Consecutively rotate about the two remaining non-local and non-opposite squares to move the local square (containing the original node) to the correct position (on the correct hexagon).

As, using this combination of automorphisms, every node can be mapped to every other node, B_4 is node-symmetric (as required).

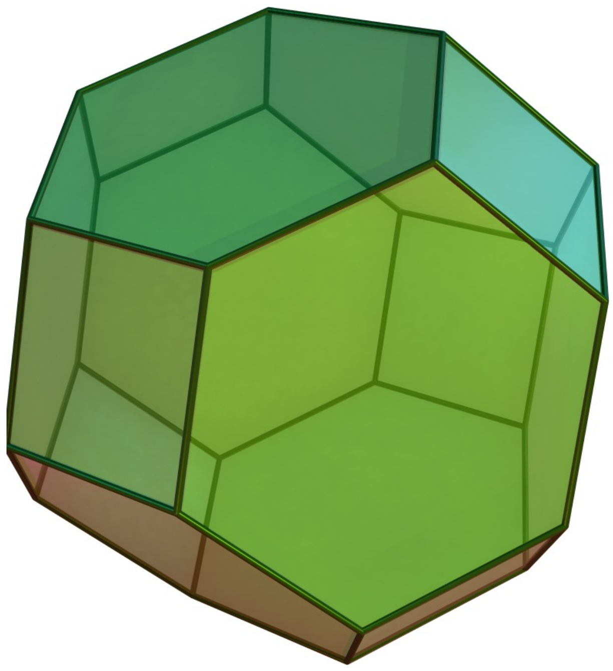
Figure 5: The truncated octahedron, image from [1].

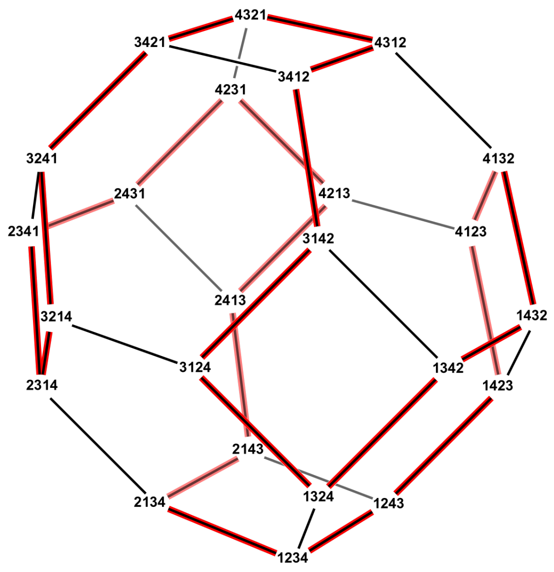

Figure 6: The order-4 Cayley graph in S_4, image from [2].

Question 5

(a)

Read the final non-root node, $H_{n-1} = (0, 0, 3)$, where H is an ordered list such that H_i is the i-th node in the cycle, and $|H| = n = 65$. On the first clock-tick, send the message bound for h_{n-1}, routing it through its preceding nodes in H. Next, read the second-to-last node $H_{n-2} = (1, 0, 3)$. On the second clock-tick, send the message bound for h_{n-1}, routing it through its preceding H. Do this for all nodes except the source, $H_i : i = n-1, n-2, ..., 3, 2$, always routing the message bound for H_i through the nodes leading up to it $(H_2, H_3, ..., H_i)$.

As a message does not need to be sent to the root node, 63 clock-ticks are required. Each channel buffer from one node to another in H has a maximal size of one unit (as one message as being either forwarded or received at any time).

(b)

Take the following mapping $\phi : K \to Q$ such that

$$\phi = \left\{(i, j, k) \to \left(\left\lfloor \frac{i}{2} \right\rfloor, \left\lfloor \frac{j}{2} \right\rfloor, \left\lfloor \frac{k}{2} \right\rfloor\right) : i, j, k \in \{0, 1, 2, 3\}\right\}$$

Conceptually, this means that in the $4 \times 4 \times 4$ lattice formed by the Q_3^4, the nodes are separated into 8 $2 \times 2 \times 2$ corner octants, each of which is assigned to one of the 8 nodes in Q_3, and the links in Q_3 represent the in and outbound channels between the octants in Q_3^4, including the wrap-around links in all dimensions. Effectively, each corner of the cube represents the corresponding eight corner cubes which form the torus.

The obvious overhead of this is that each node of Q must time-share to perform the processing and message parsing between the links of 8 nodes in K. Aside from this, the channels between the nodes in Q are used to pass every message between the octants, this consists of representing 8 links (16 channels) in K per link in Q.

To simulate the single node scatter of K in Q, the Hamiltonian cycle method used in (b) can be simulated on the virtual nodes, with messages being sent across channels Q when they are being routed to another octant. The overhead for this isn't too bad, since ϕ gives a quick calculation to convert a virtual node from K to a physical node in Q.

For a slight boost in simulation performance, as the nodes are capable of sending messages to more than one neighbour at a time, and they are aware of where the packets are getting routed ahead of time (as they have prior knowledge of H), Q can pre-route messages bound for a different octant, so as to reduce the computation time spent potentially sending and receiving the same message on the physical interconnections of Q, reducing overall load on Q while still simulating K. Additionally, this can be performed in parallel for nodes forwarding messages bound for others in Q.

Question 6

(a)

Take the following mapping $\phi : D \to C$ such that

$$\phi = \{(i, j, k, l) \to (i, j, k) : i, j, k, l \in \{0, 1\}\}$$

This means that the two nodes $(\boldsymbol{a}, 0)$ and $(\boldsymbol{a}, 1)$ of D and the link between them are simulated on node $(\boldsymbol{a})$ of C. Additionally, this means that each link in C must represent the two links in D linking $(\boldsymbol{a}, 0)$ to $(\boldsymbol{a}', 0)$ and $(\boldsymbol{a}, 1)$ to $(\boldsymbol{a}', 1)$ where $\boldsymbol{a}$ and $\boldsymbol{a}'$ differ in one bit.

As each process or C is having to simulate two processors and the corresponding links of D, the overheads of this means that S will run at most half as quickly as if it were running on D, and the load on each channel of C is expected to be doubled.

(b)

As, under the embedding in (a), each processor of C represents two processors of D running S, one can incrementally scale C by linking a new processor to one in C that is running the compatibility-layer software which simulates S for two nodes (i.e. a node that hasn't already been scaled to two nodes), and adding links to its corresponding neighbouring nodes. When linking the new processor, it should be marked as faulty / offline, this avoids stopping the execution of S as it is capable of handling one faulty node at a time. Once it is linked up and ready, the software on the original node should be switched to running the non-simulated, normal version of S, and at the same time the node marked as faulty should be brought online and started running the same version of S. To account for the additional links to the neighbours of the new node, the simulation software on the neighbours should be amended to utilise the new links (instead of the original one).

When adding further nodes, any links between their original nodes and non-original neighbours must be broken, and added back going to the new node, during which S should treat the new node as faulty until the correct corresponding links have been established and the processor has been brought online. This process can be incrementally continued until the Q_4 is formed, without the need to stop S, while only needing to swap out the simulation of S for S itself.

Part 3

Question 7

The ideal throughput is useful information because it determines the best throughput achievable by the channels in the interconnection network under the worse case within a given traffic scenario (where a bottleneck channel may limiting the network throughput). Given the channel bandwidth b and maximum channel load $\gamma_{\max} = \max_{c \in C} \gamma_c$ for channel set C, the ideal throughput is given by $\Theta_{ideal} = \frac{b}{\gamma_{\max}}$. Hence, ideal throughput scales linearly with b such that more bandwidth means more throughput, and inversely with $\gamma_{\max}$ where greater load across the bottleneck channel means lower ideal throughput. The ideal throughput is best suited to worst-case traffic scenarios as it will inform of what the bottleneck channel(s) are.

Question 8

(a)

The hypercube of dimension 8, Q_8, is a candidate topology for the interconnection network as it requires $2^8 = 256$ nodes, and $210 \leq 256 \leq 260$.

The 4-ary 4-cube, Q_4^4, can be used as an interconnection network as it requires $4^4 = 256$ nodes, and $210 \leq 256 \leq 260$. Other possible candidates such that $k \geq 3, n \geq 2$ are the 3-ary 5-cube, with $3^5 = 243$ nodes, and the 16-ary 2-cube, with $16^2 = 256$ nodes. Hence, the k-ary n-cube is a candidate topology for the interconnection network.

The cube-connected cycles of dimension 5, CCC_5, requires $2^5 \cdot 5 = 160$ nodes, the cube-connected cycles of dimension 6, CCC_6, requires $2^6 \cdot 6 = 384$ nodes, both of these lie outside the range $160 < 210 \to 260 < 384$, thus the cube connected cycles topology cannot be used for this interconnection network.

(b)

The router delay of 15ns is not enough to disrupt the frequency. Hence, under the assumption that one bit can be sent out of and received by a pin for every tick at $1GHz$, then this equates to `1Gb/s` per pin. Table 1 shows the properties of the candidate topologies mentioned above (Q_8, Q_2^{16}, Q_4^4, and Q_5^3).

Table 1: Properties of the candidate interconnection networks.

Candidate	k	n	$\|N\|$	B_C	$\|C\|$	H_{ave}	$degree$	$\frac{60}{degree}$	b	$\frac{2bB_C}{\|N\|}$	$\frac{b\|C\|}{\|N\|H_{ave}}$
Q_8	-	4	256	256	2048	4	16	3	3	6	6
16-ary 2-fly	16	2	256	64	1024	8	8	7	7	3.5	3.5
4-ary 4-fly	4	4	256	256	2048	4	16	3	3	6	6
3-ary 5-fly	3	5	243	324	2430	3.33	20	3	3	8	9

While the required values for Q_4^4 were calculated for sake of completion, this was technically unnecessary because Q_4^4 is isomorphic to Q_8.

Hence, for Q_8, $\Theta_{ideal} \leq \min\{6, 6\} = 6$`Gb/s`, for Q_2^{16}, $\Theta_{ideal} \leq \min\{3.5, 3.5\} = 3.5$`Gb/s`, for Q_4^4, $\Theta_{ideal} \leq \min\{6, 6\} = 6$`Gb/s`, and for Q_5^3, $\Theta_{ideal} \leq \min\{8, 9\} = 8$`Gb/s`).

This means that the order of preference based on maximum ideal throughput goes: (from best to worst) Q_5^3, $\{Q_8, Q_4^4\}$, and Q_2^{16}.

Question 9

Let N denote the nodes and C denote the channels in G. The tornado traffic pattern creates a path from every source node to its furthest possible (worst-case) destination node. As the network is node-symmetric, the length of the path from any source node to its worst-case destination node is the diameter of the network, denoted $H_{\max}$.

Because of this, depending on the routing, the average hop length H_{ave} is at least the diameter $H_{\max}$ (or in optimal routing it *is* the diameter). Hence, $|N|$ nodes each sending a message taking at least $H_{\max}$ hops each to reach the destination means that, on average, the total number of hops to be traversed by all messages is at least $|N|H_{\max}$. Thus, on average, at least one channel has a load of at least $\frac{|N|H_{\max}}{|C|}$, meaning $\lambda_{\max} \geq \frac{|N|H_{\max}}{|C|}$. Hence, $\Theta_{ideal} = \frac{b}{\lambda_{\max}} \leq \frac{b}{\left(\frac{|N|H_{\max}}{|C|}\right)} = \frac{b|C|}{|N|H_{\max}}$.

The upper bound on the ideal throughput under the tornado traffic pattern is therefore $\Theta_{ideal} \leq \frac{b|C|}{|N|H_{\max}}$.

Question 10

(a)

$KKK_n(N, L)$ with nodes N and links L is defined as follows

$$N = \{(i, j) : 0 \leq i, j \leq n-1, i \neq j\}$$

Each node label is a pair. The first component represents the position of the length $n-2$ cycle within the length n fully connected graph, K_n. The second component represents the position of the node within the length $n-1$ cycle. For a node to be valid, the first and second component cannot be the same.

$$\begin{aligned} L =&\{((i,j),(j,i)) \in N^2 : 0 \leq i, j \leq n-1, i \neq j\} \cup \\ &\{((i,j),(i,j+1) \in N^2 : 0 \leq i, j \leq n-1, i \notin \{j-1, j\})\} \cup \\ &\{((i,j-1),(i,j+1) \in N^2 : 0 \leq i, j \leq n-1, i = j)\} \end{aligned}$$

With all addition and subtraction $\bmod(n-1)$. The first part of L specifies the links between the cycles of length $n-2$ (effectively fully connecting the cycles). The second and third parts of L specify the links within the cycles, the first defining links between consecutively-named nodes (based on their second component), and the second defining links between nodes who's second component is two units away from it (to avoid creating invalid links between non-existent nodes (i, i) and $(i, i \pm 1)$).

(b)

The diameter of K_n is 1 for all $n > 1$, as there is a link joining all nodes. The diameter of a cycle length $n-2$ is $\lfloor \frac{n-2}{2} \rfloor$, as the shortest path between two opposite nodes must require traversal of half the cycle (whether clockwise or anticlockwise). Hence, in KKK_n, a packet travelling from one node to another must traverse at most the diameter of two length $n-2$ cycles and one K_n (or the single hop between cycles), meaning the diameter of KKK_n is at most $2\lfloor \frac{n-2}{2} \rfloor + 1 = n - 2 + 1 = n - 1$, thus $H_{\max} \leq n-1$ or $H_{\max} < n$.

Question 11

(a)

As Q_2^3 is symmetric, it can be drawn with the root $(0, 0)$ in the middle (to form a 3×3 mesh with wraparounds). Figure 7 shows Q_2^3 centered at $(0, 0)$ with the nodes isomorphically labelled using -1 instead of 2 for conceptual simplicity.

To make the spanning tree T, node $(0, 0)$ sends a message to its neighbour at () Since Q_2^3 is symmetric, this routing can be rotated about 90, 180, and 270 degrees to reach the other 6 nodes, and thus produce a spanning tree. Due to the rotational symmetry, channels from $(0, 0)$ to $(\pm 1 \bmod 3, 0)$ and $(0, \pm 1 \bmod 3)$ each have loads of two, and channels from $(2, 0)$ to $(2, 2)$, $(0, 1)$ to $(2, 1)$, $(1, 0)$ to $(1, 1)$, and $(0, 2)$ to $(1, 2)$ each have loads of 1. Figure 8 shows T using the $-1 = 2$ convention.

$$\begin{aligned} E^+(T) = \{&((0,0),(0,2),2),((0,2),(1,2),1),\\ &((0,0),(2,0),2),((2,0),(2,2),1),\\ &((0,0),(0,1),2),((0,1),(2,1),1),\\ &((0,0),(1,0),2),((1,0),(1,1),1)\} \end{aligned}$$

Where elements $c \in E^+(T)$ denote $c = (c_{source}, c_{destination}, c_{label\ or\ load})$.

Finally, T must satisfy $\sum_{c_{label} \in E^+(T)} E^+(T) \cap pos_1 = \sum_{c_{label} \in E^+(T)} E^+(T) \cap pos_2 = \sum_{c_{label} \in E^+(T)} E^+(T) \cap neg_1 = \sum_{c_{label} \in E^+(T)} E^+(T) \cap neg_2$.

$$\begin{aligned} E^+(T) \cap pos_1 =& \{((0,0),(1,0),2),((0,2),(1,2),1)\}\\ E^+(T) \cap pos_2 =& \{((0,0),(0,1),2),((1,0),(1,1),1)\}\\ E^+(T) \cap neg_1 =& \{((0,0),(2,0),2),((0,1),(2,1),1)\}\\ E^+(T) \cap neg_2 =& \{((0,0),(0,2),2),((2,0),(2,2),1)\} \end{aligned}$$

$$\begin{aligned} \sum_{c_{label} \in E^+(T)} E^+(T) \cap pos_1 =& 2 + 1 = 3\\ \sum_{c_{label} \in E^+(T)} E^+(T) \cap pos_2 =& 2 + 1 = 3\\ \sum_{c_{label} \in E^+(T)} E^+(T) \cap neg_1 =& 2 + 1 = 3\\ \sum_{c_{label} \in E^+(T)} E^+(T) \cap neg_2 =& 2 + 1 = 3 \end{aligned}$$

Hence, T satisfies the conditions.

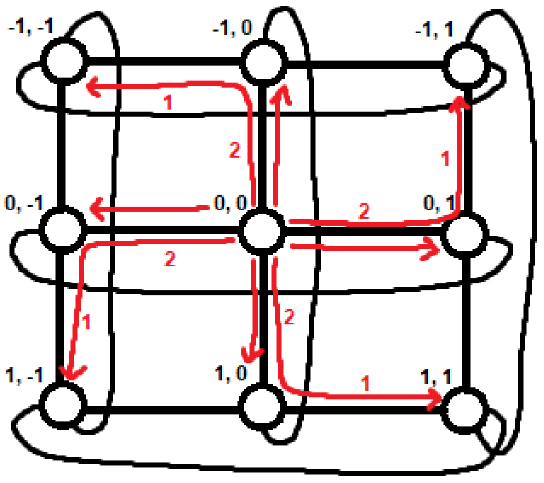

Figure 7: Q_3^2 centered at $(0,0)$ with nodes labelled using $-1 \mod 3$ instead of 2.

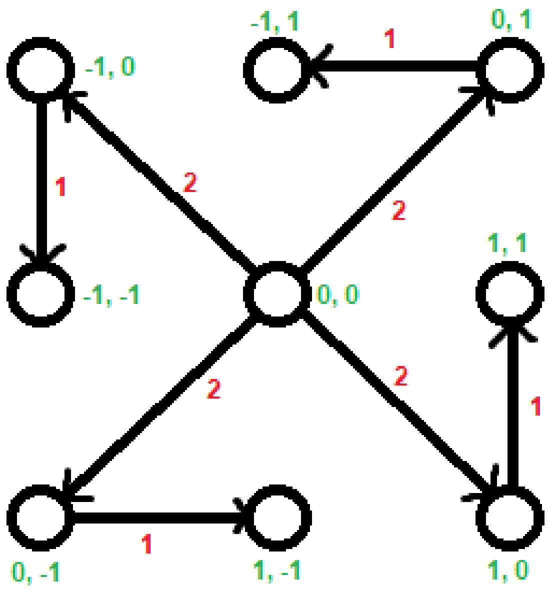

Figure 8: Spanning tree T with nodes labelled using $-1 \mod 3$ instead of 2.

(b)

By the node- and channel-symmetry of Q_2^3, there exists an automorphism that maps any node to any other node. One such automorphism is the cyclic (*single node rotation about the cycle*) $\alpha_0 = \{(a,b) \to (a+1,b) : 0 \le a,b \le 2\}$ and $\alpha_1 = \{(a,b) \to (a,b+1) : 0 \le a,b \le 2\}$ with addition modulo 3, repeated application of α allows any node and its neighbours to be interchanged with any other node and its neighbours using the 2-dimensional $\phi(n,m) = \alpha_0^n \cdot \alpha_0^m = \{(a,b) \to (a+n,b+m) : 0 \le a,b \le 2\}$ with addition modulo 2. This automorphism can also be expressed as compositions of the geometric rotation, singular use of α^0 or α^1, and reflective automorphisms.

Using the out-channels of T, $E^+(T)$, a message can be sent from the root $(0,0)$ to any other in either 1 or 2 hops, and 4 messages may be sent at a time (upper-bounded by the out-degree of Q_2^3). Using R as previously described, $\phi(s_0,s_1)$ can map any route $[(0,0),...,(i,j)]$ under R rooted at $(0,0)$ to any other route $[(0+s_0,0+s_1),...,(i+s_0,j+s_1)]$ rooted at (s_0,s_1) with addition modulo 3, allowing every node to send a message to every other node. This forms the routing algorithm R using T.

Since R is able to route from any node to any other, again using the node- and channel-symmetry of the graph, one need to only consider a single node and its in- and out-bound channels. In the all-to-all traffic pattern, every node sends a message to every other node. Consider this taking place for every node, for a node (i, j), it alone will produce an output load of 2 along $((i, j), (i \pm 1, j))$ and $((i, j), (i, j \pm 1))$. During part (a), it was demonstrated that the total loads of the channels in $E^{+}(T)$ for each direction were 3, thereby upper bounding the load on both input and output channels. From here, the third unit of load on the outbound channels originate from its neighbours, passing through (i, j), using one outbound channel each (at a load of 1 per outbound channel). Hence, the outbound channels of (i, j) have load 3. For completeness, considering the inputs to node (i, j) (though this need not be done because, by symmetry, one node's input is another node's output), the four adjacent nodes, $(i \pm 1, j)$ and $(i, j \pm 1)$, each provide a load of 2 per input channel, and one of the nodes adjacent to each of them provides an input load of 1 per input channel, for a total load of 3. Hence, all channels have load 3.

Part 4

Question 12

(a)

Claim The augmented cube AQ_n has a node degree of $\delta_n = 2n-1$ links for $n \geq 1$.

Proof Induction on n.

For the base case, $n = 1$, by definition AQ_n consists of one link joining nodes 0 and 1. Nodes 0 and 1 each have degrees $\delta_1 = 1 = 2 \cdot 1 - 1$, thus holding true the base case.

Assuming true for $n = k$, $\delta_k = 2k - 1$ for AQ_k. Now consider $n = k + 1$, we wish to show that $\delta_{k+1} = 2(k+1) - 1 = 2k + 2 - 1 = 2k + 1$ for AQ_{k+1}.

As per the definition of AQ_{k+1} two copies of AQ_k, called AQ_k^0 and AQ_k^1 (both of degree $\delta_k = 2k-1$ by the inductive assumption). One bit of value b is added to the label of $AQ_k^b, b \in 0, 1$. To create AQ_{k+1}, two links per node must now be formed between AQ_k^0 and AQ_k^1 (the usual Q_{k+1} links of hamming distance 1 between corresponding nodes, and the augmented links of hamming distance k). As two links have been added per node, their degree is increased by two, hence $\delta_{k+1} = \delta k + 2 = 2k - 1 + 2 = 2k + 1$ as required. Thus, the claim holds for all $n \geq 1$.

(b)

Claim Given AQ_n with $2n+1$ mutually node-disjoint paths joining any two nodes x and y, and any two distinct nodes $u = (u_1, u_2, ..., u_n, 0), v = (v_1, v_2, ..., v_n, 0) \in AQ_{n+1}$, there exist $2(n+1)+1$ mutually node-disjoint paths joining u and v.

Proof Another induction on n.

As AQ_n is node- and link-symmetric, the result from (a) shows that $deg_n(x) = deg_n(y) = \delta_n = 2n-1, x, y \in AQ_n$. Hence, we wish to show that any two distinct nodes $u = (u_1, u_2, ..., u_n, 0), v = (v_1, v_2, ..., v_n, 0) \in AQ_{n+1}$ have $\min\{deg_{n+1}(u), deg_{n+1}(v)\} = \min\{2n+1, 2n+1\} = 2n+1$ mutually node-disjoint paths joining them.

Now, begin the induction on n.

For the base case $n = 1$, it is clear that $AQ_1 = K_1$, which has $2 \cdot 1 - 1 = 1$ node-disjoint path.

Next, we assume that there are indeed $2n-1$ node-disjoint paths in $x, y \in AQ_n$ where $x = (u_1, u_2, ..., u_n)$ and $y = (v_1, v_2, ..., v_n)$.

First, partition AQ_{n+1} over its last dimension into $u, v \in AQ_n^0$ (by the stated definitions of u and v) and AQ_n^1). Under the inductive assumption, there are $2n-1$ mutually node-disjoint paths between u and v in AQ_n^0. The degrees $deg_{n+1}(u) = deg_{n+1}(v) = 2(n+1)-1 = 2n+1 : u, v \in AQ_{n+1}$. This provides a soft upper-bound the number of node-disjoint paths between u and v. From here, by the inductive assumption, $deg_n(x) = deg_n(y) = 2n-1 : x, y \in AQ_n$ means that x and y must use every adjacent link. Hence, going from u to x (and respectively v to y) creates a difference of $deg_n(u) - deg_n(x) = 2n+1-(2n-1) = 2$ (and respectively $deg_n(v) - deg_n(y) = 2n+1-(2n-1) = 2$) available links between AQ_n^0 and AQ_N^1.

Fortunately, by the recursive definition of AQ_{n+1}, every node $((A))$ in AQ_n^0 has exactly two links crossing the partition to AQ_n^1, these being $((\mathbf{A}, 0), (\mathbf{A}, 1))$ and $((\mathbf{A}, 0), (\bar{\mathbf{A}}, 1))$ (and vice verse).

Hence, only two mutually node disjoint paths in AQ_n^1 are required to produce the final two node-disjoint paths between u and v. The first thing to observe is that the nodes that u and v link to are $M_u = \{(\mathbf{u}, 1), (\bar{\mathbf{u}}, 1)\}, M_v = \{(\mathbf{v}, 1), (\bar{\mathbf{v}}, 1)\} \in AQ_n^1$ where $M_u \cap M_v = \emptyset$. This means we can exploit the inductive assumption recursively two form two node-disjoint paths in AQ_n^1 from $(\mathbf{u}, 1)$ to $(\mathbf{v}, 1)$ and $\bar{\mathbf{u}}, 1)$ to $\bar{\mathbf{v}}, 1)$, thus completing the remaining two node-disjoint paths between u and v.

To complete the inductive step, we sum up the mutually node-disjoint paths available to get $2n-1+2 = 2n+1 = 2(n-1)+1$ as required, which is both equal to the soft degree upper bound previously mentioned and the inductive target, thus completing the proof.

This proof was inspired by lecture 7 and checked in accordance with the proof of Proposition 4.1 in [3].

Question 13

In a 2-ary n-fly G, consider source node $0 \leq s \leq 2^{n-1}$ and destination node $0 \leq d \leq 2^{n-1}$ with a binary representation $d = (d_0, d_1, ..., d_{n-1})$ such that d_0 is the most significant bit.

Starting at s in G, move to switch node $0.\lfloor \frac{s}{2} \rfloor$.

Next, for each bit $d_u, 0 \leq u \leq n-3$ in d do the following:

- If $v < 2^{n-u-2}$ and $d_u = 0$, take the channel from switch node $u.v$ to $(u+1).v$.
- If $v \geq 2^{n-u-2}$ and $d_u = 0$, take the channel from switch node $u.v$ to $(u+1).(v + 2^{n-u-2})$.
- If $v < 2^{n-u-2}$ and $d_u = 1$, take the channel from switch node $u.v$ to $(u+1).(v - 2^{n-u-2})$.
- If $v \geq 2^{n-u-2}$ and $d_u = 1$, take the channel from switch node $u.v$ to $(u+1).v$.

Finally, take the channel from switch node $(n-1).v$ to terminal node $2v$ if $d_{n-1} = 0$, or $2v+1$ if $d_{n-1} = 1$.

From the four cases listed above, it is clear that as u increments, 2^{n-u-2} halves, meaning that when the current most significant bit is removed from consideration, $(n-u-1) \cdot 2^{n-u-2}$ switch nodes and 2^{n-u-1} potential destination nodes are also removed from potentially being in the routing. The cases ensure that based on the previous bit, the correct destination node s is not removed from consideration.

Put simply, for each bit in the destination, destination-tag routing goes along the channel from $u.v$ and takes the path that will either reduce the hamming distance between the binary representation of v and the binary representation of $\lfloor d \rfloor 2$ by one, or not increase it by one. This guarantees that by the last switch node, $v = \lfloor d \rfloor 2$, allowing the final part of the destination-tag routing to select the even or odd terminal node $2\lfloor d \rfloor 2 + d_{n-1} = d$.

Chaining this together as a summation for all iterations results in the expression $2(d_0 \cdot 2^{n-2} + d_1 \cdot 2^{n-3} + ... + d_{n-3} \cdot 2^1 + d_{n-2} \cdot 2^0) + d_{n-1} \cdot 2^0 = d_0 \cdot 2^{n-1} + d_1 \cdot 2^{n-2} + ... + d_{n-2} \cdot 2^1 + d_{n-1} \cdot 2^0 = \sum_{i=0}^{n-1} d_i \cdot 2^{n-1-i}$ which, by definition, is d as a decimal number. As there is no mention of s, this means that under this routing algorithm, any $0.a, 0 \leq a \leq 2^n - 1$ switch node will route a message to d. And hence, as the channel $(s, 0.\lfloor \frac{s}{2} \rfloor)$ is used initially, destination-tag routing yields a path for any s and d.

Question 14

According to [4], whereas switch-centric DCNs consist of servers connected by switches (such that no two servers are linked together), switch-centric DCNs consist of switches connected by servers (such that no two switches are linked together).

Server-centric DCNs are typically primarily restricted by the number of network access ports there are on each server, for most non-specialist commodity-off-the-shelf (COTS) hardware, this typically restricts each server to a maximum of two ports (*dual port*) for network communication. For routing, server-centric DCNs are able to use their general-purpose CPUs to calculate and optimise routes with useful properties such as load-balancing and fault tolerance able to be handled and optimised on-the-fly.

Switch-centric DCNs are less restricted in the number of interconnections they can serve. For routing, switch-centric DCNs typically rely on large and/or fast routing tables that are either hard-coded or periodically updated, which makes them limited in terms of dynamic DCN scalability.

An advantage of switch-centric DCNs compared to server-centric DCNs is that they, especially good ones, are capable of routing and forwarding data to and from far more switches / servers without the latency caused by processing overhead in servers (of whom only tend to have 2 ports and can thus only perform networking operations with 2 neighbours).

An advantage of server-centric DCNs compared to switch-centric DCNs is that the otherwise unutilised processing power of server CPUs can be put to work by undertaking networking / routing operations where switches are costly and can only be used for networking, providing much more flexibility with what can be processed and done compared to proprietary switch hardware/software.

As previously mentioned, a dual-port DCN is a server-centric DCN in which each server node has two network interface controller (NIC) ports, allowing links to two other nodes (server or local switch). Dual-port DCNs are important because existing (and COTS) servers typically only have one primary port and one reserve / backup port, making it a practical limitation when trying to keep costs down (like buying additional NIC hardware or specialist server motherboards).

The stellar construction for dual-port DCNs is a powerful method of allowing contemporary interconnection networks (INs) and their properties to be run in server-centric dual-port DCNs. As per [4], this is done by placing 2 server-nodes on each link in the IN and replacing the IN's nodes with hub switch-nodes connecting up the server-nodes on its original links. Routing in the stellar DCN is performed in the same way as on the IN, except messages are first passed through the switch hub to the respective server node before it can be sent down an IN link. The construction of the stellar DCN on the Q_3 IN (Figure 9) is shown in Figure 10.

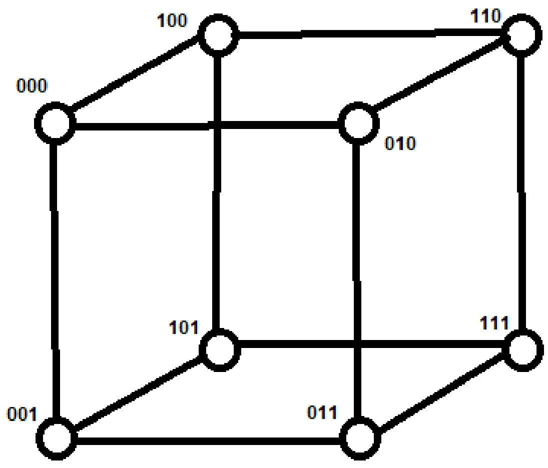

Figure 9: The 3-dimensional hypercube interconnection network Q_3.

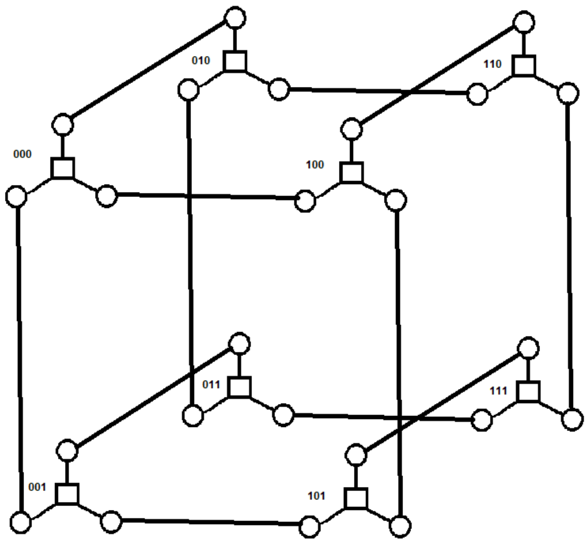

Figure 10: The 3-dimensional stellar hypercube DCN, where circle nodes represent servers and square nodes represent switches.

Part 5

Question 15

(a)

$B_n(N, L)$ is formed such that

$$N = \{(a_0, a_1, ..., a_{n-1}) \in \{0, 1, ..., n-1\}^n : i \neq j \iff a_i \neq a_j, i, j = 0, 1, ..., n-1\}$$

$$L = \{((a_0, a_1, ..., a_{n-1}), (b_0, b_1, ..., b_{n-1})) \in N^2 : a_i = b_{i+1} \wedge a_{i+1} = b_{i+1}, i = 0, 1, ..., n-2\}$$

(b)

High-level overview:

Using a modified instance of the bubble-sort algorithm, one-by-one, fix the last not-yet considered label element in the destination and perform a series of swaps in the current node to move ('bubble') the correct label item up to its destination position (where the current node starts as the source node). Every time a swap is made, add the resultant node (or corresponding link) to the routing. This ensures that valid links are made, because they correspond to a single swap in the label.

Pseudocode description:

```
1. Take input of source node $s$ and destination $d$ where $|s| = |d| = n$, and maintain a list for the routing $r$.
2. Place $s$ in $r$ as the first element and define local variable $c \leftarrow s$ to represent the current node.
3. For $j \leftarrow n-1, n-2, ..., 0$
4.     For $i \leftarrow 0, 1, ..., j-1$
5.         If $c_i = d_j$
6.             Swap the values of $c_i$ and $c_{i+1}$.
7.             Add $c$ to the end of $r$.
8.         End If
9.     End For
10. End For
11. Output routing $r$.
```

(c)

The function `bubblesort_stats(n, sources, destinations, verbose)` is provided to perform rudimentary analysis of the B_n routing algorithm. It runs the algorithm on each source destination pair provided (in this case the all-to-all traffic pattern), and collects various statistics about path lengths and loads on channels / nodes.

When run for $n = 6$ using the following statement, it outputs the proceeding results.

```
if __name__ == "__main__":
    from alltoall_generator import alltoall_traffic
    n = 6
    bubblesort_stats(n, *alltoall_traffic(n), n < 4)
```

```
n = 6
len(sources) = len(destinations) = 518400
count_ex_matches = 517680
max_path_length = 15
```

```
total_path_length = 3888000
average_path_length = 7.5
average_path_length_ex_matches = 7.510431154381084
min_channel_load = 600 (((0, 1, 2, 3, 4, 5), (0, 1, 2, 3, 5, 4)))
max_channel_load = 1368 (((0, 1, 2, 3, 4, 5), (0, 2, 1, 3, 4, 5)))
total_channel_load = 3888000
average_channel_load = 1080.0
load_channels = {600: 720, 1056: 720, 1332: 720, 1368: 720, 1044: 720}
min_node_load = 5400 ((0, 1, 2, 3, 5, 4))
max_node_load = 5400 ((0, 1, 2, 3, 5, 4))
total_node_load = 3888000
average_node_load = 5400.0
load_nodes = {5400: 720}
```

`load_channels` and `load_nodes` show the respective number of channels and nodes that have a given load. These results are shown in Figures 11 and 12 respectively. Additionally, the results show that the maximum path length is $H_{max} = 15$ and the average is $H_{ave} = 7.5$ (where the path length of a `routing` including the source is `len(routing) - 1`).

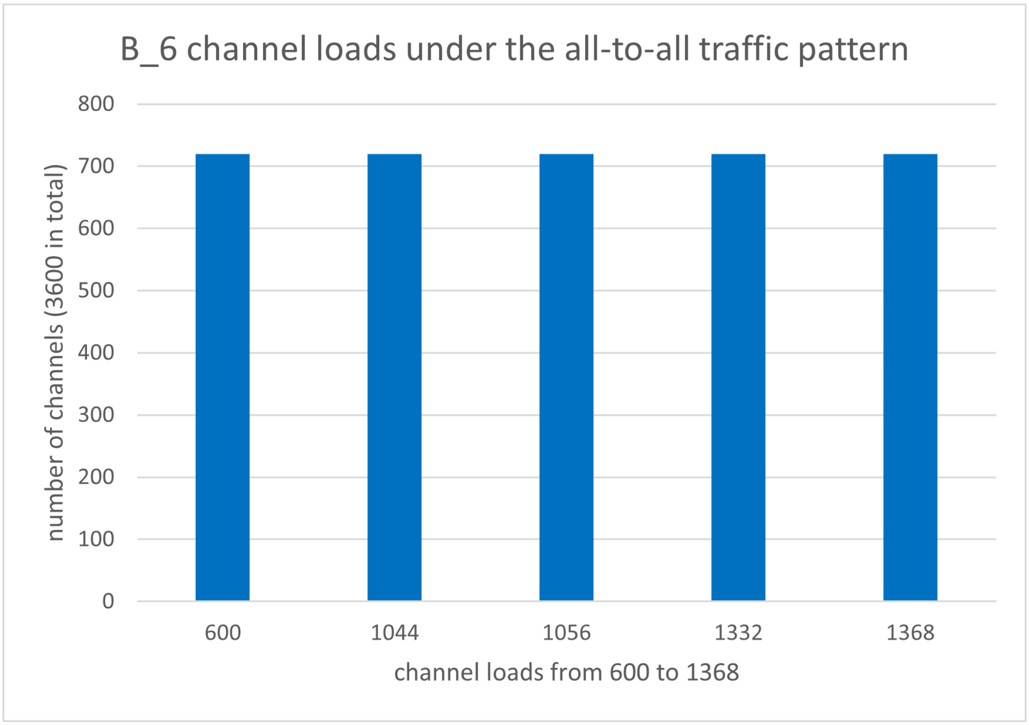

Figure 11: Channel loads of B_6 under the all-to-all traffic pattern.

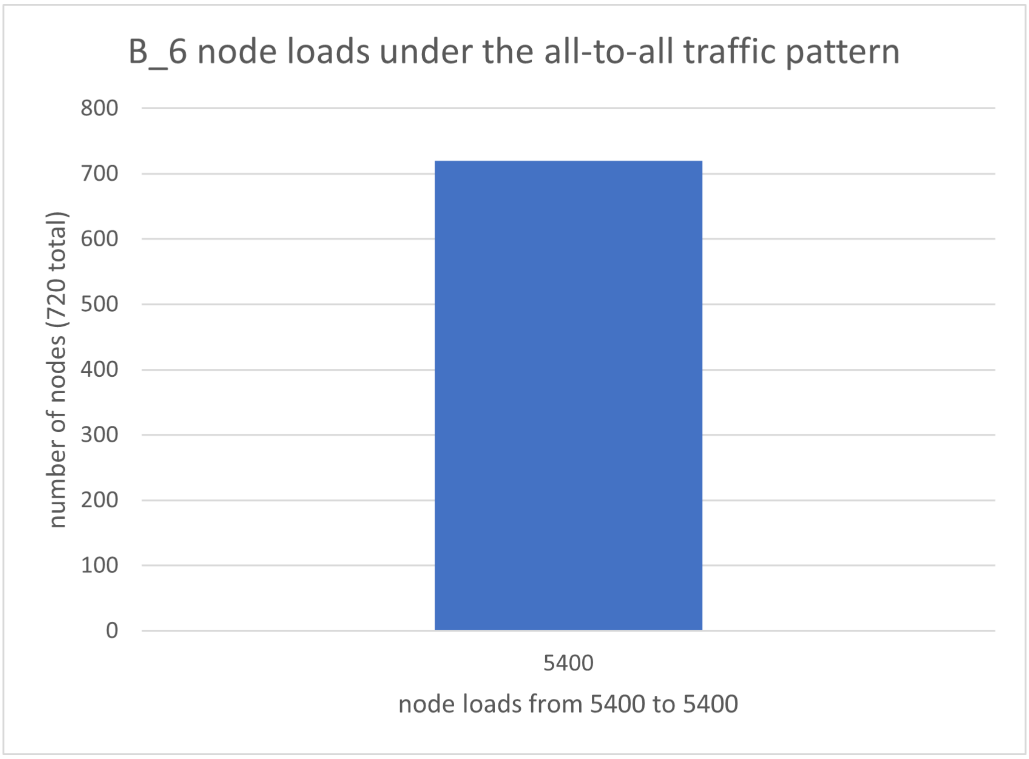

Figure 12: Node loads of B_6 under the all-to-all traffic pattern.

Question 16

`faulty_hypercube_routing` provides a fault-tolerant hypercube routing algorithm using A* search with hamming distance as the heuristic.

As a naive implementation, `faulty_hypercube_routing_dbfs` provides a fault-tolerant hypercube routing algorithm using breadth-first search (with parameter `dfs = False`) or depth-first search (with parameter `dfs = True`).

References

[1] “Truncated octahedron,” https://en.wikipedia.org/wiki/Truncated_octahedron, accessed: 2021-12-01.

[2] “Symmetric Group S_4 - Wikiversity,” https://en.wikiversity.org/wiki/Symmetric_group_S4, accessed: 2021-12-01.

[3] S. A. Choudum and V. Sunitha, “Augmented cubes,” *Networks: An International Journal*, vol. 40, no. 2, pp. 71–84, 2002.

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