Hindawi
VLSI Design
Volume 2017, Article ID 9427678, 15 pages
https://doi.org/10.1155/2017/9427678

*Research Article*

# A Framework for Scalable TSV Assignment and Selection in Three-Dimensional Networks-on-Chips

**Amir Charif,[1,2] Alexandre Coelho,[1,2] Nacer-Eddine Zergainoh,[1,2] and Michael Nicolaidis[1,2]**

[1] *TIMA, Université Grenoble Alpes, 38000 Grenoble, France*
[2] *CNRS, TIMA, 38000 Grenoble, France*

Correspondence should be addressed to Amir Charif; amir.charif@univ-grenoble-alpes.fr

Received 29 June 2017; Revised 29 September 2017; Accepted 12 November 2017; Published 11 December 2017

Academic Editor: Marcelo Lubaszewski



3D integration can greatly benefit future many-cores by enabling low-latency three-dimensional Network-on-Chip (3D-NoC) topologies. However, due to high cost, low yield, and frequent failures of Through-Silicon Via (TSV), 3D-NoCs are most likely to include only a few vertical connections, resulting in incomplete topologies that pose new challenges in terms of deadlock-free routing and TSV assignment. The routers of such networks require a way to locate the nodes that have vertical connections, commonly known as elevators, and select one of them in order to be able to reach other layers when necessary. In this paper, several alternative TSV selection strategies requiring a constant amount of configurable bits per router are introduced. Each proposed solution consists of a configuration algorithm, which provides each router with the necessary information to locate the elevators, and a routing algorithm, which uses this information at runtime to route packets to an elevator. Our algorithms are compared by simulation to highlight the advantages and disadvantages of each solution under various scenarios, and hardware synthesis results demonstrate the scalability of the proposed approach and its suitability for cost-oriented designs.

## 1. Introduction

Networks-on-Chip (NoCs) [1] have proven to be a fast and scalable replacement for buses in current and emerging many-core systems. They are today widely adopted in Chip Multiprocessors (CMPs), Multiprocessor Systems-on-Chip (MPSoCs), and even Graphics Processing Units (GPUs) [2, 3]. Moreover, the recent emergence of 3D integration can further increase the viability of Networks-on-Chip as a communication paradigm by enabling the stacking of several silicon layers and allowing for inherently low-latency three-dimensional NoC topologies (3D-NoCs) to be considered [4, 5].

Through-Silicon Via (TSV) is one of the most promising technologies that enable vertical communication between different NoC layers [6]. However, due to the high cost and low yield of TSVs [7], vertically partially connected NoCs, in which only a subset of the nodes are vertically connected, appear to be a reasonable compromise [8]. Moreover, TSVs are very likely to suffer from reliability issues [9], rendering some vertical connection unusable during runtime and further reducing the number of vertical paths in the topology.

Because such partial topologies require adequate routing algorithms to ensure correct operation and deadlock-freedom, several deadlock-free routing algorithms have already been proposed [10–12]. Regardless of which routing rules are applied, since only some of nodes are connected to TSVs, routers need a reliable way to locate the nodes that are vertically connected, commonly referred to as elevators. Both the information regarding the elevators, which is set at configuration time, and the way this information is used by the routers during runtime play a decisive role in the chip's performance.

Due to its critical importance to both performance and implementation cost, we dedicate this article to the exploration of various algorithms for elevator selection. Using Elevator-First [10] as a baseline routing algorithm, we propose a set of scalable, easy to implement elevator assignment strategies, each featuring both the information to be stored in each router and the algorithms used at configuration time

and during runtime to select the best elevator. We test and compare all of the proposed solutions in terms of performance through cycle-accurate simulation and demonstrate the scalability of our methods through hardware synthesis.

The remainder of this paper is structured as follows: A brief survey of existing elevator selection methodologies is presented in Section 2. In Section 3, the baseline NoC architecture is described. Two types of selection approaches are then presented in Sections 4 and 5. The proposed algorithms are evaluated through hardware synthesis and cycle-accurate simulation in Section 6. Section 7 concludes this work.

## 2. Related Work

A variety of routing algorithms targeting vertically partially connected 3D-NoCs have been proposed in the literature. Many of these algorithms need to follow specific rules that require TSVs to be placed in a specific manner and often have further constraints as to which TSVs can be selected during runtime. In [13], the authors propose two routing algorithms named SBSM (Source-Based Shortest Manhattan) and DBSM (Destination-Based Shortest Manhattan). SBSM selects the vertical link that is the closest to the source node, whereas DBSM selects the vertical link that is the closest to the destination. To make this possible, each router has to know the addresses of all the vertically connected nodes (elevators), which implies a significant hardware overhead. In their more recent works, the authors have introduced the Dynamic-Quadrant Partitioning algorithm [14], which uses a constant number of bits per router to select an elevator. This makes the solution more interesting in terms of implementation cost. However, the algorithm can only take an elevator located in the northeast quadrant.

The East-then-West (ETW) algorithm [11, 14] is a routing algorithm that requires that at least one TSV be placed in the eastmost column in order to guarantee reachability. Due to the routing rules, the set of elevators that can be selected is constrained by the position of the destination. Each router needs to know the location of 3 different elevators: 2 nearest elevators in the east and west directions and 1 elevator in the eastmost column, in order to select the correct elevator based on the destination's position. Consequently, each router stores 3 node addresses, which limits the scalability of the routing logic.

By contrast with the aforementioned algorithms, Elevator-First [10] does not impose any constraints on the placement or the selection of elevators. By using Elevator-First as a baseline algorithm, we are therefore able to develop generic selection strategies that are not limited by the TSV placement strategy or any algorithm-specific constraints.

We identify two approaches to elevator selection for Elevator-First in the literature.

The first approach was introduced as part of the original Elevator-First proposal in [15]. The authors propose selecting an elevator for each router at configuration time (offline) and storing its address in a register. When a packet reaches a new layer, the current router prepends a new header to the packet, containing the address of its selected elevator. This mechanism has the major advantage of being generic and compatible with many offline selection algorithms. However, since complete node addresses need to be stored in the routers, the size of configurable data grows with the network size. In this paper, we want to show the different selection strategies that are possible using a constant amount of bits per router.

Similarly to our method, the second approach aims at addressing the scalability issues of the original Elevator-First and is part of the LBDR3D framework [16]. The authors use a limited amount of configurable bits in each router, named Vertical Bits, to point to the nearest elevator. The nearest elevator is selected offline based on the Manhattan Distance, and when several elevators with an equal distance from a given router exist, ties are broken randomly. Unfortunately, this specification suffers from a few issues that have not been addressed. First, because different routers may point to different elevators, there can be cases where one router forwards a packet in the direction of its own selected elevator and where the next router forwards it to another direction towards its own elevator, in such a way that the two directions form an illegal turn, leading to potential deadlocks. In this work, when we consider the Manhattan Distance for elevator selection in Section 4, we provide offline and online solutions to this critical problem. Second, in [16], no proof of reachability was provided, and additional input signals were introduced to prevent packets from entering livelocks. In this paper, we provide a universal formal proof of reachability for all Manhattan Distance-Based selection approaches, removing the need for any additional signals to ensure reachability.

Neither Elevator-First nor LBDR3D can take the packets' destination into account when selecting an elevator, as the elevators are selected offline in both approaches. This can heavily limit the level of adaptability of the routing solution in some cases. In addition to the Manhattan Distance-based algorithms, we also propose a method for selecting an elevator online based on the destination, while still using the exact same amount of information as the distance-driven approaches.

## 3. Target Architecture

*3.1. NoC Architecture.* We consider a network comprised of several 2D mesh layers connected vertically using TSV, as shown in Figure 1. Only a subset of the routers is vertically connected, and these routers are referred to as "elevators." The problem of finding the best placement of TSVs at different layers has already been studied [17] and is beyond the scope of this paper. The algorithms proposed throughout this paper are compatible with any placement strategy. Moreover, the TSV pillars need not be placed in the same positions across all layers.

*3.2. Routing.* To provide a deadlock-free routing solution, we rely on the routing rules of Elevator-First [10]. That is, the network is virtually portioned into two virtual networks using two separate input FIFOs (a.k.a. virtual channels) in each planar port. Packets heading to an upper layer are injected in the first virtual channel, whereas packets heading down are routed in the second virtual channel. As per Elevator-First,

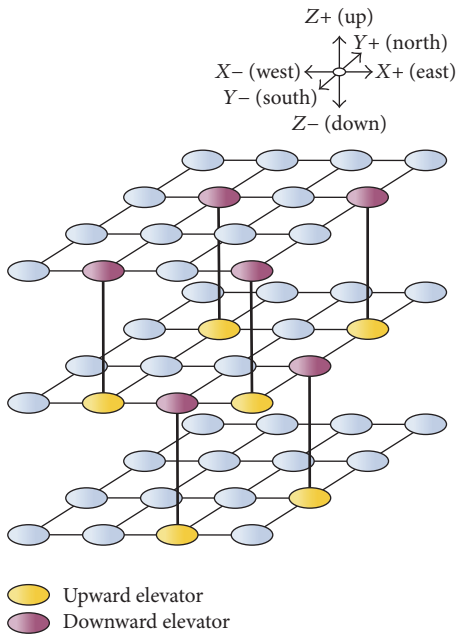


Figure 1: Overview of the partially connected 3D-NoC Topology.

routing within each layer is performed using a deadlock-free 2D routing algorithm. For the sake of illustration, the *XY* algorithm is assumed throughout this article.

Despite using the same deadlock avoidance technique, our routing methodology is different from the one described in [10, 15]. In [10], each router stores the address of the nearest elevator and every time the packet reaches a new layer, a new header containing the elevator's address is prepended to the packet. For distributed operation and scalability, our approach does not involve storing node addresses. Instead, each router includes a fixed number of configurable bits named Elevator Location Bits, which contain information about the location of elevators. These bits can be reconfigured at any time to reflect the new state of the network upon the occurrence of TSV failures. The number of these bits is independent of the size of the topology. In addition, these bits are never inserted in the packet's header but are used directly by the route computation logic to guide packets towards an elevator. The route computation logic can be generically described as in Algorithm 1. As is the case for all conventional router architectures, route computation starts by comparing the router's address to that of the destination. The result is a vector of signal bits that we call compare bits. If the destination is on the same layer, the simple logic of *XY* is used to determine the next output port. If the destination is on a different layer and the current router is an elevator, then route towards the destination layer (Line (12)). If the destination is in a different layer and the current router is not an elevator, then use the Elevator Location Bits and the compare bits to route towards an elevator (Line (10)).

Our focus in the rest of the paper is to answer the following questions: (i) What to put in the Elevator Location Bits? (ii) How to use these bits online (Algorithm 1, Line (10)). Several approaches offering different levels of complexity and performance are explored.

## 4. Manhattan Distance-Based Elevator Selection

One possible solution to our problem simply consists in choosing an elevator that is located as close as possible to the current router. The idea behind this approach is to minimize the time spent searching for an elevator and to quickly reach the destination layer. This is the criterion of selection that many other works have been adopting. In this section, we present three efficient algorithms that exploit the properties of Manhattan Distance to minimize the information required for locating the nearest elevators, while still guaranteeing reachability.

*4.1. Elevator Location Bits.* To reach one of the nearest TSV pillars, only 8 bits of information per router are sufficient. Let elevator be a 4-bit vector stored within a router and (Elevator.North, Elevator.East, Elevator.South, and Elevator.West) its four configurable bits. This vector uses the same encoding as the compare bits described previously. That is, each bit is set so as to indicate whether the selected elevator is in the given direction. For instance, if the offline configuration algorithm selects an elevator located northeast to the current router, elevator will be set to $(1, 1, 0, 0)$. Each router needs to store two such bit vectors, one for the upward elevator and one for the downward elevator. This encoding allows for the efficient routing algorithm implementation presented in Algorithm 2. Here, elevator is set to either the upward or downward elevator according to the destination.

*4.2. Safe Selection Algorithm (MD-Safe).* Given this encoding, all that the configuration algorithm has to do is select one elevator for each router. Here again, several approaches are possible. One thing to take into consideration is the fact that this encoding allows different routers to point to different nearest elevators, and consequently one router that forwards a packet in the direction of its nearest elevator cannot guarantee that it will reach that same elevator after traversing the next hops. The first approach that we propose is to set these bits in such a way that all routers along one path point to the same elevator. This can be achieved using Algorithm 3 for each layer. Here, we iterate through each elevator in turn and check if it is the nearest elevator to every node in the layer. Even if the distance from some node to the new elevator is the same as its previously assigned nearest elevator, the new elevator is still preferred. This ensures that starting from any initial node *A*, which is pointing to elevator node *E*, routing in the direction of *E* reaches another node *B* that points to the same nearest elevator *E*. If *B* had a nearest elevator node *F* different from *E*, then according to Algorithm 3, *F* would also be the nearest elevator to *A*. Therefore, our algorithm inherently guarantees that packets always reach their intended elevator. This approach also has the advantage of being independent of the planar routing algorithm; that is, this offline algorithm is compatible with any online routing function. For instance, in [18], we have used this selection approach in combination with a routing algorithm that uses the adaptive Negative-First [19] algorithm for intralayer routing.

```
Input:
    Elevator (Elevator location bits)
Output:
    Direction (Output port)
(1) Variable: Compare (Comparison bits)
(2) Compare.East = current.X < dest.X
(3) Compare.West = current.X > dest.X
(4) Compare.North = current.Y < dest.Y
(5) Compare.South = current.Y > dest.Y
(6) if current.Z ≠ dest.Z then                ▷ different layer
(7)     if current.isElevator then
(8)         Direction = (current.Z < dest.Z?Up : Dn)
(9)     else
(10)        Use Elevator and Compare to find an elevator
(11)    end if
(12) else                                     ▷ XY logic
(13)    if Compare.East then
(14)        Direction = East
(15)    else if Compare.West then
(16)        Direction = West
(17)    else if Compare.North then
(18)        Direction = North
(19)    else if Compare.South then
(20)        Direction = South
(21)    else
(22)        Direction = Local
(23)    end if
(24) end if
```

Algorithm 1: Route computation logic (generic).

```
Input:
    Elevator (Elevator location bits)
Output:
    Direction (Output port)
(1) Variable: Compare (Comparison bits)
(2) Compare.East = current.X < dest.X
(3) Compare.West = current.X > dest.X
(4) Compare.North = current.Y < dest.Y
(5) Compare.South = current.Y > dest.Y
(6) if current.Z ≠ dest.Z then                ▷ seek elevator
(7)     if current.isElevator then
(8)         Direction = (current.Z < dest.Z?Up : Dn)
(9)     else
(10)        Direction = XY(Elevator)
(11)    end if
(12) else
(13)    Direction = XY(Compare)
(14) end if
```

Algorithm 2: Route computation with MD-based elevator selection.

*4.3. Randomized Selection Algorithm (MD-Random).* While the safe selection algorithm has the interesting property of achieving consensus among several routers about where the nearest elevator is, it may not offer the best load balancing and elevator utilization, as many nodes will attempt to reach the exact same elevator at once. Intuitively, better performance can be achieved by selecting a random elevator among several nearest elevators, as the load would be more uniformly distributed among TSVs.

Figure 2 illustrates the difference when using both algorithms. Notice how MD-safe forces consensus by making several routers point to the same elevator, whereas MD-random provides a better distribution.

One challenging aspect of such a randomized approach is that it may cause packets to violate the routing rules, resulting in deadlocks.

Consider the example shown in Figure 3, where a packet originates at node $A$ and needs to take an elevator. In this example, two elevators, $E1$ and $E2$, are available. They are assigned to nodes $A$ and $B$, respectively. At router $A$, the packet takes the north direction to reach $E1$. However, at node $B$, it will take the west turn to reach $E2$ following the $XY$ algorithm. By taking the west turn after the north turn, the algorithm has already violated the rules of $XY$. We propose two methods to alleviate this issue.

The first approach consists in rewriting Algorithm 2 in such a way that $Y$ to $X$ turns cannot be made. The alternative routing algorithm is presented in Algorithm 4, where input_direction indicates the direction from which the packet has arrived. The idea behind this method is straightforward: if a packet in search for an elevator is received at the north (or south) port, then it forcibly has an elevator in the south

```
Output:
    Elevator[LayerNodes] (Elevator location bits)
(1)  Variable: Dist[LayerNodes] (Distance to closest elevator)
(2)  for all node i do
(3)      Initialize Dist[i] to infinity
(4)      Initialize Elevator[i] to all zeros
(5)  end for
(6)  for all elevator (xE, yE) do
(7)      for all node i of coord (x, y) do
(8)          if |xE − x| + |yE − y| ≤ Dist[i] then
(9)              Dist[i] = |xE − x| + |yE − y|
(10)             Elevator[i].North = (yE > y)
(11)             Elevator[i].South = (yE < y)
(12)             Elevator[i].West = (xE < x)
(13)             Elevator[i].East = (xE > x)
(14)         end if
(15)     end for
(16) end for
```

Algorithm 3: Setting the elevator bits (safe).

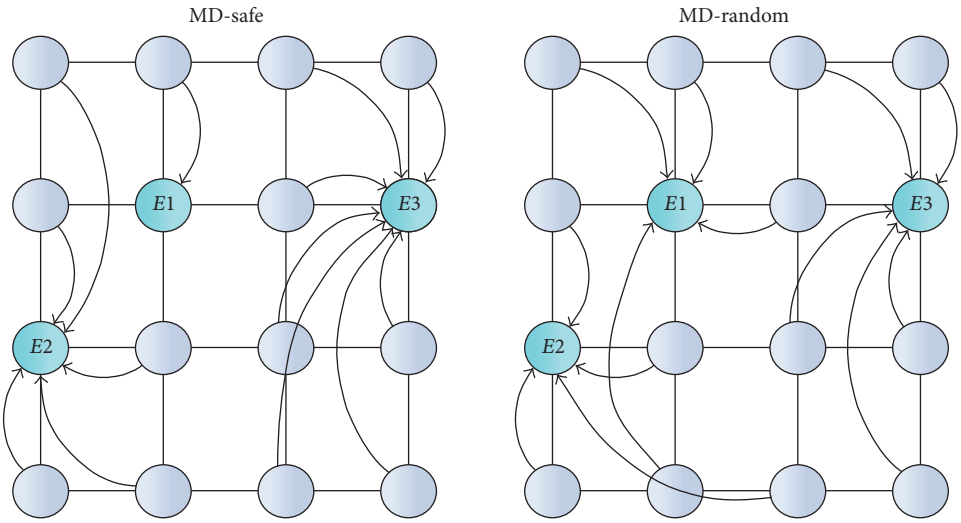


Figure 2: Example TSV assignment when using MD-safe versus MD-random.

(or north) direction; otherwise the previous router would not have forwarded it following the $Y$ dimension. This means that it is enough to make sure that packets traveling along the $y$-axis keep going in the same direction until an elevator is eventually reached. While simple, the main drawback of this approach is that it heavily depends on the $XY$ algorithm and very hardly adapts to other algorithms. In fact, in the case of an adaptive routing algorithm, it is not possible for the current router to infer which elevator was intended for the packet simply from the direction it has taken last, as it may have been only one of the possible directions available at the previous router.

A less rigid approach consists in maintaining the original online routing algorithm and preventing deadlock scenarios at the offline selection stage. We propose a method that is compatible with $XY$ as well as the three deadlock-free adaptive turn models [19]. One property of all of these algorithms is that they impose an order on the traversal of physical channels. For instance, in the West-First turn model, the east, north, and south directions are taken last. In the North-Last turn model, the north direction is taken last. In the $XY$ algorithm, north and south are taken last. The idea is to exploit this property during the elevator selection process, by giving precedence to the elevators that can be reached using only the last directions of the given routing algorithm.

Since we are working with the $XY$ algorithm, the selection algorithm can be written as in Algorithm 5. By prioritizing the nearest elevators that are on the same column, we ensure that a packet only takes east or west when there are no closest elevators in the same column. Once the $Y$ dimension is taken, all subsequent routers will agree that there is a nearest elevator on the same column as per Algorithm 5, and there

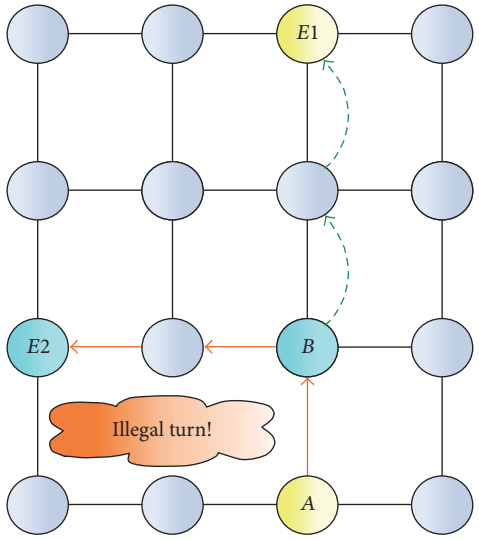


Figure 3: Illustration of a potential deadlock scenario.

**Input:**
Elevator (Elevator location bits)
**Output:**
Direction (Output port)
(1) Variables: Compare (Comparison bits)
(2) *Compare.East* = *current.X* < *dest.X*
(3) *Compare.West* = *current.X* > *dest.X*
(4) *Compare.North* = *current.Y* < *dest.Y*
(5) *Compare.South* = *current.Y* > *dest.Y*
(6) **if** *current.Z* ≠ *dest.Z* **then**
(7) **if** *current.isElevator* **then**
(8) *Direction* = (*current.Z* < *dest.Z*?*Up* : *Dn*)
(9) **else if** *input_direction* == *North* **then**
(10) *Direction* = *South*
(11) **else if** *input_direction* == *South* **then**
(12) *Direction* = *North*
(13) **else**
(14) *Direction* = *XY*(*Elevator*)
(15) **end if**
(16) **else**
(17) *Direction* = *XY*(*Compare*)
(18) **end if**

Algorithm 4: Route computation for online deadlock-freedom (random-online).

will therefore not be a need to take the $X$ dimension again. This effectively removes any risk of deadlocks.

The algorithm operates as follows: For every node $i$ in a given plane, elevators are sorted according to their Manhattan Distance from $i$. Then only the nodes that are at minimum distance are considered (Line (7)). The rest of the algorithm breaks the ties between these minimum distance elevators as described previously. First, the algorithm checks whether there are elevators on the same column as the current node. If there are no such elevators (Line (10)), then an elevator is selected randomly from the set of minimum distance

**Output:**
Elevator[LayerNodes] (Elevator location bits)
(1) **for all** node $i$ **do**
(2) Initialize Elevator[$i$] to all zeros
(3) **end for**
(4) **for all** node $i$ of coord $(x, y)$ **do**
(5) $E$ = List of all elevators
(6) Sort $E$ By distance from $i$
(7) $Min = e \in E / distance(e, i) == distance(E[0], i)$
(8) $SameCol = (x', y') \in Min / x' == x$
(9) **if** SameCol is empty **then**
(10) $(xE, yE)$ = random elevator from Min
(11) **else**
(12) $(xE, yE)$ = random elevator from SameCol
(13) **end if**
(14) $Elevator[i].North = (yE > y)$
(15) $Elevator[i].South = (yE < y)$
(16) $Elevator[i].West = (xE < x)$
(17) $Elevator[i].East = (xE > x)$
(18) **end for**

Algorithm 5: Setting the elevator bits for offline deadlock-freedom (random-offline).

elevators. Otherwise (Line (12)), an elevator is selected randomly from the set of minimum distance elevators on the same column. The elevator bits are then set to point to the chosen elevator (Lines (14) to (17)).

A more general form of the algorithm, which is not tied to a specific routing algorithm, is also presented in Algorithm 6. This generic algorithm takes the set of the last directions of the planar algorithm *LastDirSet* as an input. The last direction set is defined as follows.

*Definition 1* (last direction set). Let $D$ be the set of all planar directions in a mesh network, such that $D = \{West, South, East, North\}$. A deadlock-free planar routing algorithm can be defined as a list $A$ of subsets of $D$ [20]. Let $A = \{D_0, D_1, \ldots, D_n\}$. The last direction set $L$ of algorithm $A$ is simply the last element $(D_n)$ of $A$.

For instance, the West-First routing algorithm can be written as

$$A = [\{West\}, \{North, South, East\}]. \quad (1)$$

The last direction set of the West-First algorithm is therefore

$$L = \{North, South, East\}. \quad (2)$$

After obtaining the set of minimum distance elevators (Line (7)), we first determine the set of directions that need to be used to reach every elevator. For instance, if elevator $e$ is at the east of the current node, then the east direction is added to the list of required directions $Dir$ (Lines (11)–(13)). If the set of directions is part of the last direction set (defined previously), then the elevator is added to a prioritized set named *Last* (Lines (23)–(25)). Finally, the algorithm selects

```
Input:
    LastDirSet (The last directions set of the routing algorithm)
Output:
    Elevator[LayerNodes] (Elevator location bits)
(1)  for all node i do
(2)      Initialize Elevator[i] to all zeros
(3)  end for
(4)  for all node i of coord (x, y) do
(5)      E = List of all elevators
(6)      Sort E By distance from i
(7)      Min = e ∈ E/distance(e, i) == distance(E[0], i)
(8)      Last = ∅
(9)      for all e(x', y') ∈ Min do
(10)         Dir = ∅
(11)         if x' > x then
(12)             add East to Dir
(13)         end if
(14)         if x' < x then
(15)             add West to Dir
(16)         end if
(17)         if y' > y then
(18)             add North to Dir
(19)         end if
(20)         if y' < y then
(21)             add South to Dir
(22)         end if
(23)         if Dir ⊂ LastDirSet then
(24)             add e to Last
(25)         end if
(26)     end for
(27)     if Last is empty then
(28)         (xE, yE) = random elevator from Min
(29)     else
(30)         (xE, yE) = random elevator from Last
(31)     end if
(32)     Elevator[i].North = (yE > y)
(33)     Elevator[i].South = (yE < y)
(34)     Elevator[i].West = (xE < x)
(35)     Elevator[i].East = (xE > x)
(36) end for
```

Algorithm 6: Generic algorithm for setting the elevator bits for offline deadlock-freedom.

one of the elevators in this prioritized set *Last*, if any, or any random elevator from the minimum distance set otherwise.

The genericity of this algorithm makes it compatible with any routing solution.

Another challenging aspect of randomized elevator assignment is to ensure that packets eventually reach an elevator. In what follows, we provide an elaborate proof of reachability and livelock-freedom by using the properties of the Manhattan Distance.

*4.4. Proof of Reachability for Manhattan Distance-Based Approaches.* Because packets needing to reach a different layer may traverse routers that point to different elevators as per our selection algorithms, it is necessary to make sure that packets are always able to reach an elevator; that is, they are never led to a dead end and are never able to fluctuate between different nodes indefinitely.

While related works introduce extra signals to prevent packet looping at runtime [9], we provide a formal proof of reachability showing that packets are bound to reach an elevator regardless of the criteria used to select one elevator among the nearest ones. We further show that routing from any node to the final elevator is always done following the minimal distance.

**Theorem 2** (elevator reachability). *If each router forwards a packet one hop closer to one of its nearest elevators, then the packet will eventually reach an elevator.*

*Proof.* Let $(X_c, Y_c)$ be the coordinates of the current router $C$ in a given routing scenario. Let $(X_{ec}, Y_{ec})$ be the coordinates

of the elevator $E_c$ selected by the offline algorithm for router $C$. The Manhattan Distance between node $C$ and its elevator $E_c$ is defined as follows: $\mathrm{MD}(C, E_c) = |X_c - X_{ec}| + |Y_c - Y_{ec}|$.

We know that the current router will forward packets to a next node $N\ (X_n, Y_n)$ so as to get closer to $E_c$.

By definition, we have

$$\mathrm{MD}\left(N, E_c\right) = \mathrm{MD}\left(C, E_c\right) - 1. \tag{3}$$

That is, router $N$ is closer to $E_c$ than $C$. Now let $E_n$ denote the elevator selected by the offline algorithm for $N$, and let $(X_{en}, Y_{en})$ be its coordinates. Because $E_c$ was selected by the offline algorithm as the elevator of $C$, we know that $E_n$ cannot be closer to $C$ than $E_c$, as otherwise $E_n$ would have been selected as the nearest elevator instead. This means that

$$\mathrm{MD}\left(C, E_c\right) \leq \mathrm{MD}\left(C, E_n\right). \tag{4}$$

The same applies to the selection of $E_n$ for $N$:

$$\mathrm{MD}\left(N, E_n\right) \leq \mathrm{MD}\left(N, E_c\right). \tag{5}$$

By combining (3) and (5), we obtain

$$\mathrm{MD}\left(N, E_n\right) \leq \mathrm{MD}\left(C, E_c\right) - 1. \tag{6}$$

This is an important property, as it shows that the distance between a node and its own selected elevator decreases at every hop. By recurrence, this implies that the distance will eventually reach 0, thereby proving that packets always reach an elevator. □

**Theorem 3** (minimality). *When seeking an elevator, the path a packet takes from any node to the final elevator is a minimal path.*

*Proof.* First, we show the distance between a node and its own elevator decreases by exactly 1 at every traversed hop.

Let us assume that there is a node $C$, with elevator $E_c$, that forwards a packet to a next hop $N$, with elevator $E_n$, such that

$$\mathrm{MD}\left(N, E_n\right) < \mathrm{MD}\left(C, E_c\right) - 1. \tag{7}$$

We know that node $C$ is able to reach elevator $E_n$ in $\mathrm{MD}(N, E_n)$ hops, plus 1 hop from $C$ to $N$. And from (7), we know that the distance from $C$ to its own elevator $E_c$ is greater than $\mathrm{MD}(N, E_n) + 1$. In other words, $E_n$ is closer to $C$ than $E_c$, which contradicts with (4).

Consequently, we obtain the following equation from (6):

$$\mathrm{MD}\left(N, E_n\right) = \mathrm{MD}\left(C, E_c\right) - 1. \tag{8}$$

Let $F(X_f, Y_f)$ be the finally reached elevator in a given routing scenario. Assuming nonminimal routing, the list of visited nodes from the source to $F$ must include two nodes $P$ and $Q$, such that $P$ was visited before $Q$ and routing from $Q$ to $F$ was done following minimal distance and

$$\mathrm{MD}\left(P, F\right) = \mathrm{MD}\left(Q, F\right). \tag{9}$$

Assuming $H$ hops were visited between $P$ and $Q$, we have from (8) that

$$\mathrm{MD}\left(P, E_p\right) = \mathrm{MD}\left(Q, E_q\right) + H. \tag{10}$$

Because routing from $Q$ to $F$ was done following minimal distance, the number of hops from $Q$ to $F$ is $\mathrm{MD}(Q, F)$; also, from (8) we know that this also corresponds to $\mathrm{MD}(Q, E_q)$. That is,

$$\mathrm{MD}\left(Q, F\right) = \mathrm{MD}\left(Q, E_q\right). \tag{11}$$

From (9), (10), and (11), we can write

$$\mathrm{MD}\left(P, E_p\right) = \mathrm{MD}\left(P, F\right) + H. \tag{12}$$

This means that $F$ is closer to $P$ than $E_p$ which again contradicts with our initial assumption that $E_p$ is the closest elevator of $P$. Therefore, routing from a source node to the final elevator is always done following the minimum distance. □

*4.5. Resilience to Runtime Failures.* An important result of the proof of reachability is that a packet reaches an elevator regardless of which of the nearest elevators is assigned to each node. This has a major implication in terms of fault-tolerance. If the system is able to detect a TSV failure and reconfigure the elevator bits, in the routers that were pointing to the failing elevator, to point to a different nearest elevator at runtime, then no packets need to be rerouted.

## 5. Optimistic Elevator Selection

The goal of the MD-based selection algorithms presented in the previous section was to minimize the distance between a source node and the selected elevator. While this is a reasonable option most of the time, it can perform poorly in various scenarios. The main reason is that the position of the final destination of the packets is never taken into account while routing a packet towards its elevator. As an example, let us consider the example shown in Figure 4. Here, the packet has originated at node $S$ and is destined for node $D$ located in a different layer. Using the previously defined algorithms, the packet is routed to the nearest elevator E1, drifting away from the destination, before reaching the destination layer. The total hop count from source to destination could have been greatly reduced had the packet taken elevator E2.

In this section, we introduce another type of selection called Optimistic Elevator Selection. In this approach, routers attempt to reduce the distance to an elevator and to the final destination simultaneously.

*5.1. Elevator Location Bits.* The exact same amount of configuration bits is required for the optimistic selection approach as the MD-based approach, thereby maintaining scalability. Here again, each router stores two 4-bit vectors (north, east, south, and west). However, the meaning of these bits differs from the previous specification. Instead of pointing to a specific elevator location, these bits act as a compass that vaguely indicates the presence of any elevators in the given directions. The north and south bits are set if there is at least one elevator in the same column to the north or to the south, respectively. The east and west, on the other hand, are used to indicate the existence of any elevator in the east or west

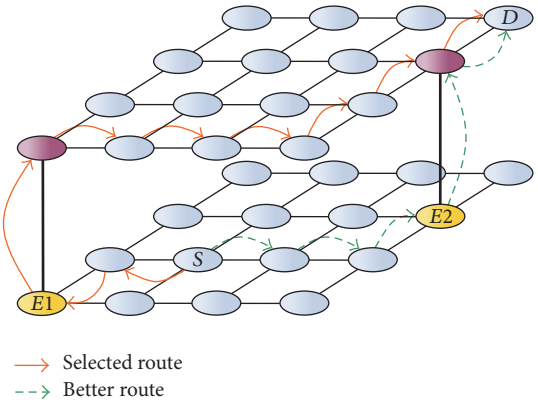


Figure 4: Example of an inefficient route when using MD-based algorithms.

**Output:**
Elevator[LayerNodes] (Elevator location bits)
(1) **for all** node $i$ **do**
(2) Initialize Elevator[$i$] to all zeros
(3) **end for**
(4) **for all** elevator ($xE$, $yE$) **do**
(5) **for all** node $i$ of coord ($x$, $y$) **do**
(6) $Elevator[i].North \mid = (yE > y \&\& xE == x)$
(7) $Elevator[i].South \mid = (yE < y \&\& xE == x)$
(8) $Elevator[i].West \mid = (xE < x)$
(9) $Elevator[i].East \mid = (xE > x)$
(10) **end for**
(11) **end for**

Algorithm 7: Setting the elevator bits (optimistic).

directions, not necessarily on the same row as the current router. That is, east is set if at least one elevator exists in the east, northeast, or southeast directions. The algorithm used to set these bits is described in Algorithm 7.

As illustrated in Figure 5, unlike the MD-based selection approaches, here the nodes are not assigned a particular elevator. Node $A$ only knows that elevators exist at both east and west of it and that no elevators are present on its column. It does not know the location of the elevators. Node $B$ has two elevators on the same column and also elevators in the east and west. Note that elevators on the east and west need not be on the same row.

*5.2. Routing Algorithm (Optimistic).* Because the selection of an elevator now accounts for the destination's position, most of the selection complexity must be transferred to the online route computation algorithm, that is, the hardware. Of course, Algorithm 2 can no longer be used. Instead we replace it by Algorithm 8. It should be noted that input_direction is assumed to be a generation variable; therefore, the test on the input direction is not performed at runtime but is processed at generation time. This means that in hardware each input port will include a different combinational logic

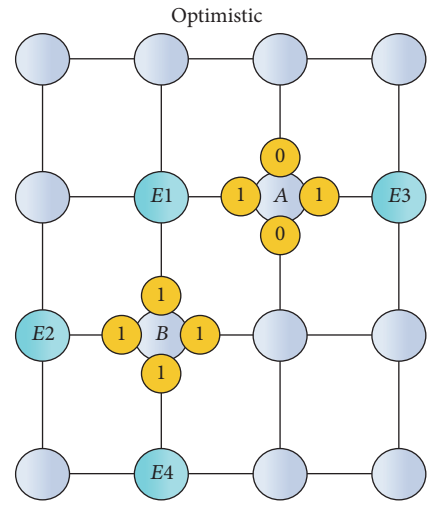


Figure 5: Elevator Location Bits for the optimistic selection algorithm.

for this algorithm. As can be seen, the logic is still quite simple.

The routing process can be described as follows. First route along the $X$ dimension while trying to get closer to the destination as long as possible. If the column of the destination was reached, route along the $Y$ dimension while trying to get closer to the destination. If the destination was exceeded following the $X$ dimension, keep going in the same direction until a column having an elevator is reached. If the destination is exceeded in the $Y$ dimension, then keep going in the same direction until an elevator is reached.

A few examples are presented in Figure 6 to illustrate the operation of the optimistic algorithm. In the first scenario, node $S1$ wants to send a message to node $D1$. If the MD-based approach were used, then $S1$ would transmit the packet to the elevator located to its east, as it is the nearest one. However, in the case of optimistic selection, packets are routed closer to the destination whenever possible, so it is forwarded west. Once the same column as the destination is reached, the packet takes an elevator on this column. Note that this path is a minimal one. If the closest elevator were selected, the packet would make two extra hops to reach the destination. This is the ideal case for optimistic selection, wherein elevators are located between the source and the destination. In the case of $S2$ and $D2$, the destination is already located on the same column, but no elevators are present on this column. In this case the packet goes in search for an elevator further from the destination. Here, elevators were available in the east direction, so an east move was made.

The last scenario ($S3$ to $D3$) shows a case where optimistic selection does not offer the best route. This is often the case when few elevators are available. Even though an elevator was available on $S3$'s column, Algorithm 8 requires that the packet moves towards the destination to the west, since elevators are available to the west. However, when the destination's column is reached, no elevators are available. In this case, the packet continues going in the same direction (west), until the column containing the elevators is found.

```
Input:
    Elevator (Elevator location bits)
    Cmp (Comparison bits)
Output:
    Direction (Output port)
(1)  if current.Z ≠ dest.Z then
(2)      if current.isElevator then
(3)          Direction = (current.Z < dest.Z?Up : Dn)
(4)      else
(5)          if input_direction == East then
(6)              if Elevator.West&Cmp.West then
(7)                  Direction = West
(8)              else if Elevator.North&Cmp.North then
(9)                  Direction = North
(10)             else if Elevator.South&Cmp.South then
(11)                 Direction = South
(12)             else if Elevator.North then
(13)                 Direction = North
(14)             else if Elevator.South then
(15)                 Direction = South
(16)             else                          ▷ the elevator is West
(17)                 Direction = West
(18)             end if
(19)         else if input_direction == West then
(20)             Same as East (replace West by East)
(21)         else if input_direction == North then
(22)             Direction = South
(23)         else if input_direction == South then
(24)             Direction = North
(25)         else          ▷ have not engaged in direction yet
(26)             if Elevator.West&Cmp.West then
(27)                 Direction = West
(28)             else if Elevator.East&Cmp.East then
(29)                 Direction = East
(30)             else if Elevator.North&Cmp.North then
(31)                 Direction = North
(32)             else if Elevator.South&Cmp.South then
(33)                 Direction = South
(34)             else if Elevator.North then
(35)                 Direction = North
(36)             else if Elevator.South then
(37)                 Direction = South
(38)             else if Direction.West then
(39)                 Direction = West
(40)             else
(41)                 Direction = East
(42)             end if
(43)         end if
(44)     end if
(45) else
(46)     Direction = XY(Cmp)
(47) end if
```

Algorithm 8: Route computation (optimistic).

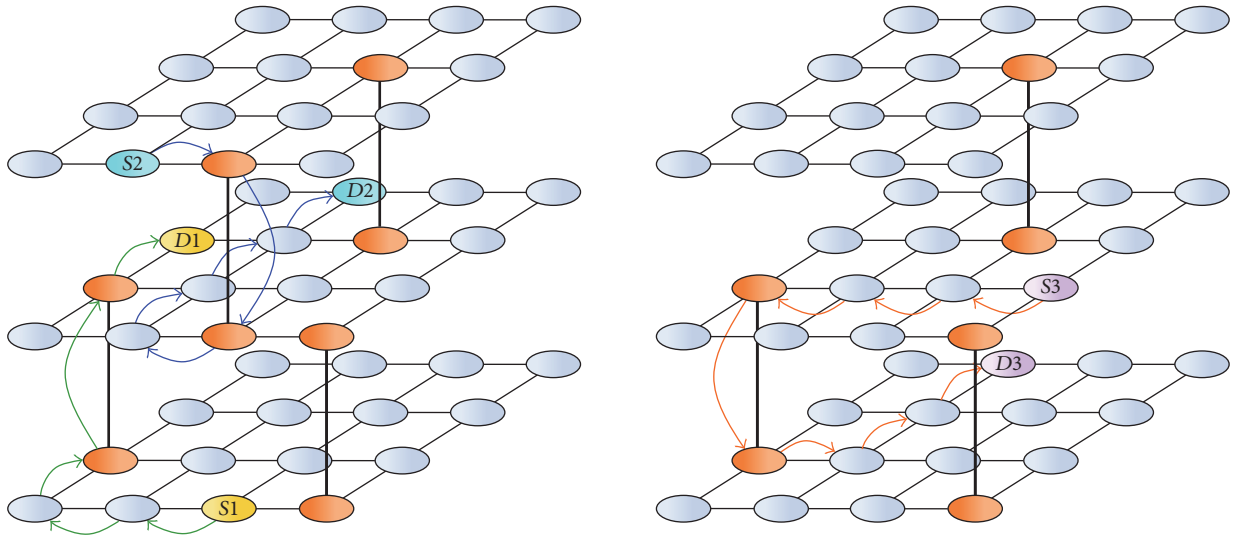


Figure 6: Routing scenarios for the optimistic selection approach.

Because routing is still performed following the *XY* rules, our algorithm has no impact on deadlock-freedom. Moreover, because the routing algorithm never selects a direction unless the configuration bits indicate the presence of an elevator, we also guarantee that the algorithm is always capable of finding an elevator.

## 6. Experimental Results

After exploring various strategies for elevator selection, we are now going to evaluate and compare them to have a clear idea of the scenarios under which each solution is the most appropriate. Two aspects of our algorithms are evaluated: hardware implementation cost and network performance.

*6.1. Hardware Synthesis Results.* To estimate the area and power costs of the proposed solutions, we have implemented a 3D router's Route Computation Unit (RCU) in SystemVerilog. The RCU takes as an input the flits coming from all the router's input channels and outputs the route computation result, that is, next output port, for each input channel. It is worth reminding that a 3D router includes one local port (packets coming from the tile), two vertical ports (up and down), and 4 planar ports (east, west, south, and north), such that each planar port includes 2 virtual channels.

For new packets, that is, when a head flit is read from an input channel at a given cycle, a new route is computed and stored in a register. The body and tail flits of the same packets will simply be routed to the same port.

We have implemented both the MD-based (Algorithm 4) and optimistic (Algorithm 8) routing algorithms described in this paper. As a reference, we have also implemented the RCU of the original Elevator-First algorithm, as described in [15]. It differs from our architecture in that the local and vertical ports generate a temporary header that includes the address of the selected elevator. The logic used to create and output this temporary header followed by the original flits was implemented in the RCU. However, the logic used to remove this header is performed on the output side, so the RCU simply computes a signal that indicates whether this header should be written or not.

The RCU also stores the configurable information required for locating the elevator nodes, that is, the elevator bits for our algorithms and the full coordinates of the selected elevator for Elevator-First.

The RCU is synthesized using Synopsys Design Compiler tool with the NanGate Open Cell 45 nm Library [21] and a power supply of 1 V. We performed two syntheses in order to evaluate first the minimum area and then the maximum operating frequency. In the first synthesis, we set the same operating frequency for all designs to 1 GHz and configure the parameters to achieve the minimum area overall and power consumption. For the second synthesis, we set up the tool to achieve the maximum operating frequency based on the critical path delay. The results for both syntheses are summarized in Table 1 for different network sizes.

First, it is interesting to compare the three algorithms for a layer size of $4 \times 4$, because these dimensions require 4 bits to address each elevator, which means that the size of configurable data in both Elevator-First and the proposed algorithms is identical. Here, it can be seen that the area of Elevator-First is slightly larger than the MD-based proposed algorithm, mainly due to the temporary header logic. However, note that the optimistic algorithm is more complex than Elevator-First. This is because although Elevator-First requires extra logic in the local and vertical ports, the presence of a temporary header actually simplifies the routing logic in the planar input channels. By contrast, the optimistic algorithm uses a more complex routing logic at the east and west ports. This is reflected by the 0.7% decrease in the maximum operating frequency.

When the network size increases, all the algorithms grow in size, as the destination and current addresses are getting larger. However, what is interesting to see is that the

Table 1: Hardware synthesis results.

| Size | Area[1] ($\mu m^2$) | | | Power[1] ($\mu W$) | | | Max. freq (MHz) | | |
|---|---|---|---|---|---|---|---|---|---|
| $(x, y, z)$ | Elevator-First | MD-based | Optimistic | Elevator-First | MD-based | Optimistic | Elevator-First | MD-based | Optimistic |
| $4 \times 4 \times 4$ | 975 | 955 | 1115 | 299 | 277 | 338 | 3709 | 3709 | 3684 |
| $8 \times 8 \times 4$ | 1173 | 1091 | 1230 | 346 | 312 | 367 | 3709 | 3709 | 3684 |
| $16 \times 16 \times 4$ | 1350 | 1170 | 1324 | 397 | 335 | 394 | 3708 | 3701 | 3684 |
| $24 \times 24 \times 4$ | 1562 | 1302 | 1450 | 445 | 363 | 424 | 3668 | 3701 | 3684 |

[1] An operating frequency of 1 GHz was used to obtain the area and the power consumption.

Table 2: Simulation parameters.

| Parameters | Value(s) |
|---|---|
| Buffers/VC | 4 |
| Flits/packet | 5 |
| Traffic pattern | Uniform, complement, shuffle |
| TSV density | 75%, 50%, 25%, 12.5% |
| Sim. duration | 100000 cycles |
| Iterations | 50 |
| 2D routing | *XY* |

size of Elevator-First grows much faster than the proposed approaches, because in addition to the compare logic the amount of configurable data, which consists of full elevator addresses, also increases. We can see that the size of Elevator-First increases by 60% when going from $4 \times 4$ to $24 \times 24$ layer size, whereas the area increase for the MD-based and optimistic algorithms is of 36.33% and 30%, respectively. In addition, we can see that the operating frequency of Elevator-First decreases by 1.2% while the other two algorithms remain almost unaffected. This shows a better scalability of the proposed approaches to large tier sizes.

*6.2. Performance Evaluation.* We are now going to test the proposed algorithms by simulation to understand how they perform with respect to each other. To this end, we use a custom cycle-accurate parallel Network-on-Chip simulator written in CUDA C [22]. The router microarchitecture, including all the proposed routing algorithms, is modeled in great detail and the network is consistently tested for incorrect behavior or potential deadlocks. The network parameters used for simulations are summarized in Table 2. TSV density corresponds to the proportion of elevators in each layer. All layers are supposed to have the same density; however, the placement of TSVs is different from one layer to the other. TSVs are placed randomly at each iteration.

The performance metric we consider is the average packet latency, which is the average elapsed time between the queuing of a packet in the network interface and the reception of its tail flit at the destination network interface. Simulations are performed on two network sizes: a 128-node network with two $8 \times 8$ layers and a 256-node network with four $8 \times 8$ layers. The goal is to evaluate the impact of the layer count on various algorithms. We present the results in Figures 7 and 8.

We first examine the latency in a network with two layers (Figure 7). The first observation that can be made is that with the optimistic algorithm the network saturates much slower than the other algorithms in most scenarios. The only case where it saturates quicker than the other algorithms is when the number of available elevators is very small, which reduces the chances of finding an elevator on the way to the destination. The effect of low TSV density on the optimistic algorithm is even more severe under complement traffic, in which all the nodes send the packets to another layer.

In shuffle traffic, where many nodes communicate within the same layer, we see that the optimistic algorithm maintains a better performance than other approaches even if only 12.5% of the nodes are vertically connected. If the system is designed in such a way that vertical communications are minimal, the optimistic routing algorithm is clearly the best option to adopt. Among the MD-based approaches, we can see that MD-random-online consistently delivers the best performance. This was expected because selection between several nearest elevators offline is performed in a fully randomized manner, whereas MD-safe and MD-random-offline both pose some deterministic constraints on the selection.

Notice that the zero load latency for all the approaches is almost the same, which indicates that the performance improvement observed for the optimistic algorithm is mainly due to the better load distribution among elevators.

We now consider the performance under a 4-layer network (Figure 8). While the optimistic algorithm still performs better under uniform and shuffle traffic, it now yields very poor performance under complement traffic (heavy interlayer communication), even under high TSV densities. This is due to the fact that when packets fail to find an optimal elevator in the first layer, they will also be penalized in subsequent layers as well. In other words, even if packets are likely to find an elevator on their path to the destination in their original layer, they are less likely to also find one in every layer along the path. This suggests that a hybrid solution, in which optimistic routing is performed only one layer away from the destination layer, can be a better compromise in scenarios where nodes often need to communicate with farther tiers.

## 7. Conclusions

Targeting 3D-NoCs with partial vertical connections, we have introduced a framework that routing algorithms can use to locate and select elevators using only a constant number of bits per router. Different ways of configuring these bits and using them during the routing process were explored.

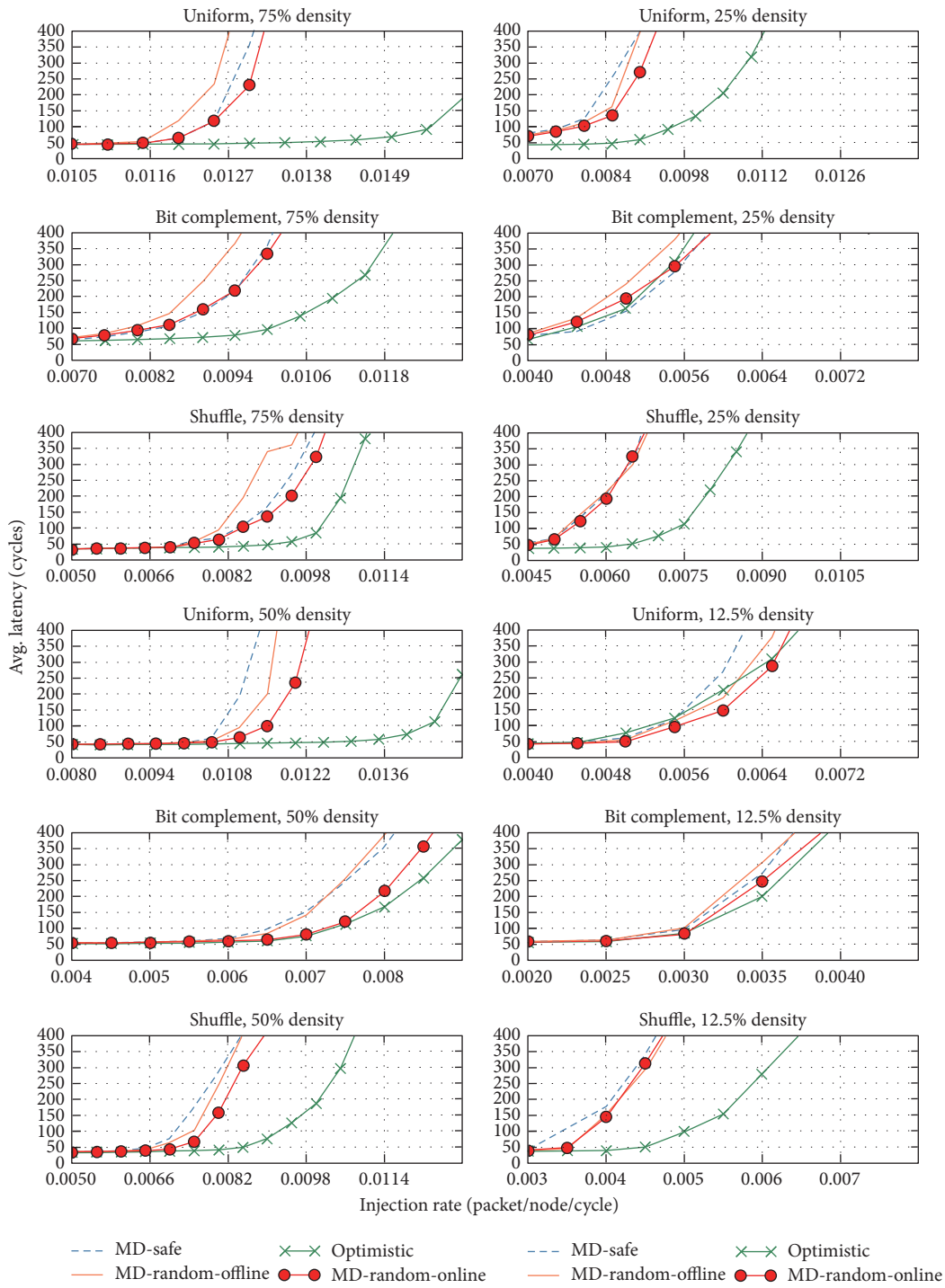


Figure 7: Average packet latency for an $8 \times 8 \times 2$ NoC.

The first type of selection is based on the Manhattan Distance (MD). Here, the elevator bits are set to point to the direction of one of the elevators located at minimum Manhattan Distance from the current router. Although MD-based selection is already adopted in the literature, we have solved many concerns to which no satisfactory and general solutions were provided. First, we have provided a proof that shows that selection algorithms based on the Manhattan Distance guarantee reachability. That is, even if a packet traverses routers that point to different elevators, it is guaranteed to eventually reach an elevator, without any alteration to the original routing algorithm. This is an important proof, as it not only removes the need to explicitly address dead-ends at runtime, but also increases the freedom of choice between several nearest elevators, resulting in higher resilience to runtime failures. Second, we have

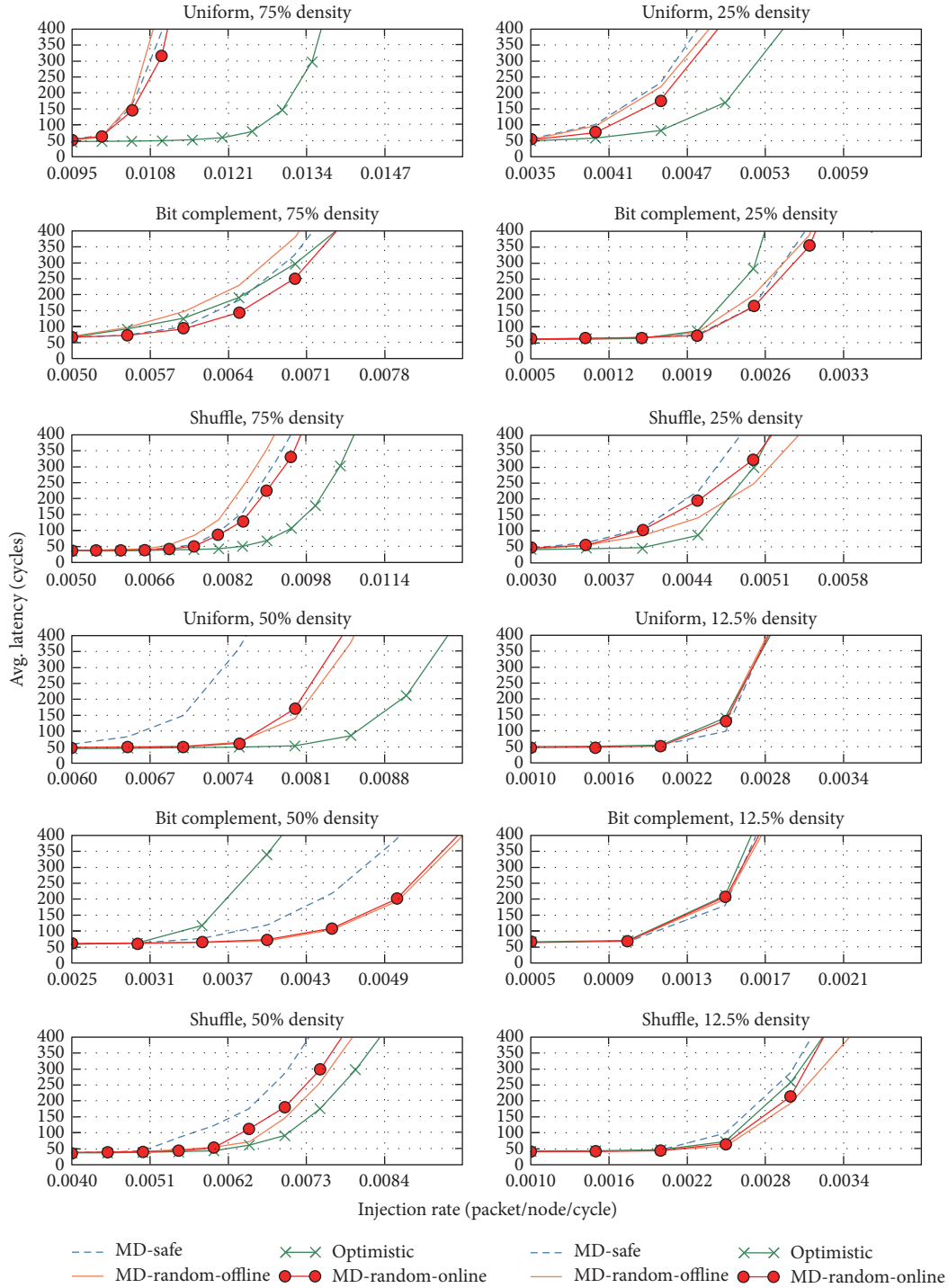


Figure 8: Average packet latency for an $8 \times 8 \times 4$ NoC.

shown that the distributed selection of elevators was not inherently deadlock-free. We have subsequently presented three selection strategies that alleviate this issue. MD-safe sets the Elevator Location Bits in such a way that routers along any given path have the same vision of the nearest elevator's location. The algorithm is simple but fails to achieve any sort of load balancing among TSVs. MD-random-offline selects among the nearest elevators randomly so as to better balance the load. However, it gives priority to elevators in a given set of directions depending on the routing algorithm in use, such that deadlock situations cannot be reached. MD-safe and MD-random-offline solve deadlocks fully at the configuration stage and require no specific modification to the hardware, making them compatible with any planar routing algorithm. To provide an even better load balancing across elevators, we have proposed MD-random-online. It

selects an elevator offline in a fully randomized manner, but the routing algorithm is modified to ensure that routing rules are not violated. We have shown that this algorithm performs better than the two offline approaches, but its implementation is heavily dependent on the routing algorithm.

The second type of selection that we have introduced improves upon the existing solutions by taking the destination into account. Here, instead of pointing to a specific elevator, the bits are set to indicate the existence of elevators in a given direction. The final elevator is determined online according to the destination. Although it uses a slightly more complex routing logic than the MD-based algorithms, it dramatically improves performance in various situations. All the algorithms that we proposed require 8 bits per router, regardless of the network size, making them highly scalable, as we have shown through the hardware synthesis results.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## References


[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 684–689, June 2001.

[2] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-effective on-chip networks for manycore accelerators," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '10)*, pp. 421–432, IEEE Computer Society, December 2010.

[3] D. Wentzlaff, P. Griffin, H. Hoffmann et al., "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[4] B. S. Feero and P. P. Pande, "Networks-on-chip in a three-dimensional environment: a performance evaluation," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 32–45, 2009.

[5] V. F. Pavlidis and E. G. Friedman, "3-D topologies for networks-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 10, pp. 1081–1090, 2007.

[6] W. R. Davis, J. Wilson, S. Mick et al., "Demystifying 3D ICs: the pros and cons of going vertical," *IEEE Design & Test of Computers*, vol. 22, no. 6, pp. 498–510, 2005.

[7] L. Benini, "3d-mpsocs: architectural and design technology outlook," in *Keynote presentation at 7th International Forum on ApplicationSpecific MultiProcessor SoC*, 2008.

[8] A. Bartzas, N. Skalis, K. Siozios, and D. Soudris, "Exploration of alternative topologies for application-specific 3d networks-on-chip," in *Proceedings of the WASP*, 2007.

[9] A. Eghbal, P. M. Yaghini, N. Bagherzadeh, and M. Khayambashi, "Analytical fault tolerance assessment and metrics for TSV-based 3D network-on-chip," *IEEE Transactions on Computers*, vol. 64, no. 12, pp. 3591–3604, 2015.

[10] F. Dubois, A. Sheibanyrad, F. Ptrot, and M. Bahmani, "Elevator-first: a deadlock-free distributed routing algorithm for vertically partially connected 3D-NoCs," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 609–615, 2013.

[11] R. Salamat, M. Ebrahimi, and N. Bagherzadeh, "An adaptive, low restrictive and fault resilient routing algorithm for 3D Network-on-Chip," in *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP '15)*, pp. 392–395, IEEE, March 2015.

[12] H. Ying, K. Hofmann, and T. Hollstein, "Dynamic quadrant partitioning adaptive routing algorithm for irregular reduced vertical link density topology 3-Dimensional Network-on-Chips," in *Proceedings of the 2014 International Conference on High Performance Computing and Simulation (HPCS '14)*, pp. 516–522, IEEE, July 2014.

[13] H. Ying, A. Jaiswal, T. Hollstein, and K. Hofmann, "Deadlock-free generic routing algorithms for 3-dimensional networks-on-chip with reduced vertical link density topologies," *Journal of Systems Architecture*, vol. 59, no. 7, pp. 528–542, 2013.

[14] R. Salamat, M. Khayambashi, M. Ebrahimi, and N. Bagherzadeh, "A resilient routing algorithm with formal reliability analysis for partially connected 3D-NoCs," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3265–3279, 2016.

[15] M. Bahmani, A. Sheibanyrad, F. Pétrot, F. Dubois, and P. Durante, "A 3D-NoC router implementation exploiting vertically-partially-connected topologies," in *Proceedings of the 2012 IEEE Computer Society Annual Symposium on VLSI (ISVLSI '12)*, pp. 9–14, IEEE, August 2012.

[16] B. Niazmand, S. P. Azad, J. Flich, J. Raik, G. Jervan, and T. Hollstein, "Logic-based implementation of fault-tolerant routing in 3D network-on-chips," in *Proceedings of the 10th IEEE/ACM International Symposium on Networks-on-Chip (NOCS '16)*, pp. 1–8, IEEE, September 2016.

[17] S. Foroutan, A. Sheibanyrad, and F. Pétrot, "Assignment of vertical-links to routers in vertically-partially-connected 3-D-NoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 8, pp. 1208–1218, 2014.

[18] A. Charif, N. Zergainoh, A. Coelho, and M. Nicolaidis, "Rout3D: a lightweight adaptive routing algorithm for tolerating faulty vertical links in 3D-NoCs," in *Proceedings of the 2017 22nd IEEE European Test Symposium (ETS '17)*, pp. 1–6, IEEE, Limassol, Cyprus, May 2017.

[19] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, pp. 278–287, 1992.

[20] M. Ebrahimi and M. Daneshtalab, "Ebda: a new theory on design and verification of deadlock-free interconnection networks," *ISCA*, pp. 1–13, 2017.

[21] Nangate, "Nangate open cell library 45nm," 2017, http://www.nangate.com/?page_id=2325.

[22] A. Charif, A. Coelho, N.-E. Zergainoh, and M. Nicolaidis, "Detailed and highly parallelizable cycle-accurate network-on-chip simulation on GPGPU," in *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC '17)*, pp. 672–677, January 2017.