

Article

Comparative Analysis of Lightweight Arbiters for Mesh NoCs

Mikhail Malikov, Mikhail Romashikhin, Evgeny Lezhnev, Roman Solovyev and Aleksandr Romanov *

HSE University, 101000 Moscow, Russia; mmalikov@hse.ru (M.M.); romashikhin.m.y@hse.ru (M.R.);
elezhnev@hse.ru (E.L.); rsolovev@hse.ru (R.S.)

* Correspondence: a.romanov@hse.ru

Abstract

One of the key mechanisms for ensuring quality of service (QoS) is arbitration, which determines the order of access to shared resources of a network-on-chip (NoC). This paper provides a comparative analysis of lightweight arbiters that consider the number of hops from the source address and the remaining hop distance to the destination address. The study was performed using the Noxim simulator for networks with a mesh topology. Various traffic patterns, routing algorithms, and network congestion levels were considered. The results show that hopcount-max is the most effective arbiter on average, providing the best results in half of the considered routing scenarios. It achieves the highest throughput under medium and high load and improves throughput over round-robin by 30% under random traffic and by 15% under shuffle traffic. Under transpose traffic, hopcount-min and distance-min reduce average delay by 60% relative to round-robin. Round-robin remains preferable in terms of maximum delay under light load. The results obtained can be used in the design of lightweight arbiters and QoS optimization in promising NoCs.

Keywords: network-on-chip; quality of service; NoC arbiter; NoC router; routing algorithm; traffic pattern

1. Introduction

Modern multiprocessor systems-on-chip (MPSoCs) are characterized by the dense integration of computing cores, accelerators, memory subsystems, and peripheral interfaces that interact with each other via NoC [1]. The use of a network for transferring data between computing cores eliminates the main problem of classical buses: they allow only one data transfer at a time, which can block all other transfers in the system [2]. Packets in an NoC can be routed along different routes; so, it is possible to transfer data between multiple source–destination pairs at the same time [3].

The essential components of an NoC are routers and channels, interconnected according to the chosen topology. The routers are connected to each other via communication channels, and the computing cores are attached to the routers [3]. Routers are designed to calculate routes and send incoming packets to the next routers or to the cores. When the network is heavily loaded and there are many incoming packets, routers also determine the order in which packets are given access to the output ports [4].

As the size and heterogeneity of MPSoCs grow, requirements for bandwidth, latency, and interconnection stability become critical, especially for real-world applications. Under these conditions, ensuring Quality of Service (QoS) in an NoC becomes a

Academic Editor: Paolo Crippa

Received: 12 September 2026

Revised: 20 October 2026

Accepted: 20 October 2026

Published: 23 October 2026

Copyright: © 2026 by the authors.
Licensee MDPI, Basel, Switzerland.
This article is an open access article
distributed under the terms and
conditions of the [Creative Commons
Attribution \(CC BY\) license](https://creativecommons.org/licenses/by/4.0/).

necessary architectural mechanism [5].

Ensuring QoS in NoCs means managing the shared use of network resources (channels, buffers) to maintain specified characteristics for different flows. The key NoC parameters are throughput and delay, which must meet the requirements of individual applications or traffic classes [6]. However, implementing QoS under conditions of limited hardware resources and high network competition is a non-trivial task that requires a compromise between implementation complexity, overhead costs, and overall network performance [7].

One option for ensuring QoS is arbitration. Arbitration is the mechanism that determines the order in which multiple contending packets are granted access to shared network resources, such as input buffers, crossbar switches, or output channels within a router. Since several packets may simultaneously request the same output port or communication link, the arbiter resolves these conflicts according to a predefined policy.

Arbitration is performed locally at each router and can operate on either flits or packets, depending on the specific router implementation. The chosen arbitration strategy has a direct impact on key network performance metrics, including latency, throughput, and fault tolerance. In particular, it governs how bandwidth is distributed among competing flows and how long packets may wait in buffers before being forwarded [8].

Classic round-robin-based approaches may be insufficient to meet the diverse QoS requirements and performance demands of modern NoCs. Consequently, various arbitration algorithms capable of considering traffic characteristics, network status, and other parameters have been actively developed in recent years. However, existing schemes based on packet characteristics are either complex to implement or rely on probabilistic weights, and they only consider the distance from the source. In this work, we introduce a novel arbitration criterion—the remaining hop distance to the destination—and evaluate it alongside a simplified deterministic hop-count-based policy. We systematically evaluate lightweight arbitration policies based on hop count and remaining hop distance, considering both minimum- and maximum-priority variants. While these metrics are individually simple, their comparative assessment across a wide range of NoC operating conditions has not been performed before. By focusing on lightweight hardware solutions, we aim to improve router operation without incurring excessive area and power overhead. The impact of different approaches on network characteristics is analyzed, and their advantages, limitations, and areas of applicability are discussed.

2. Overview of Existing Solutions

To choose the optimal arbitration scheme, it is important to consider the main types of existing solutions. They can be divided into several groups based on the key principles of their operation.

The basis for most simple and fair arbiters is provided by the schemes with cyclic priority (Round-Robin, RR). A round-robin arbiter provides a fair arbitration mechanism where the request that is granted is given the lowest priority for the subsequent arbitration cycle [9]. The main advantage of the RR-based arbiters is that they ensure that no active input is left unserved, and that there is no starvation.

More complex systems that require high efficiency rely on arbiters capable of considering network status and traffic parameters. These algorithms can analyze packet characteristics, assign static or dynamic priorities, and adapt to the current load to optimize overall network performance.

This section provides an overview of all the major well-known arbiters to provide the necessary context for evaluating new solutions.

2.1. Round-Robin-Based Arbiters

One of the first algorithms developed were iterative algorithms with the grant–accept model (Figure 1). The first algorithm based on this model—PIM (Parallel Iterative Matching)—was presented in [10]. Iterations are performed as follows:

1. All unassigned inputs send request signals to all required outputs;
2. Each output randomly selects one request and sends it a grant signal;
3. An input that receives multiple grant signals also randomly accepts one of them.

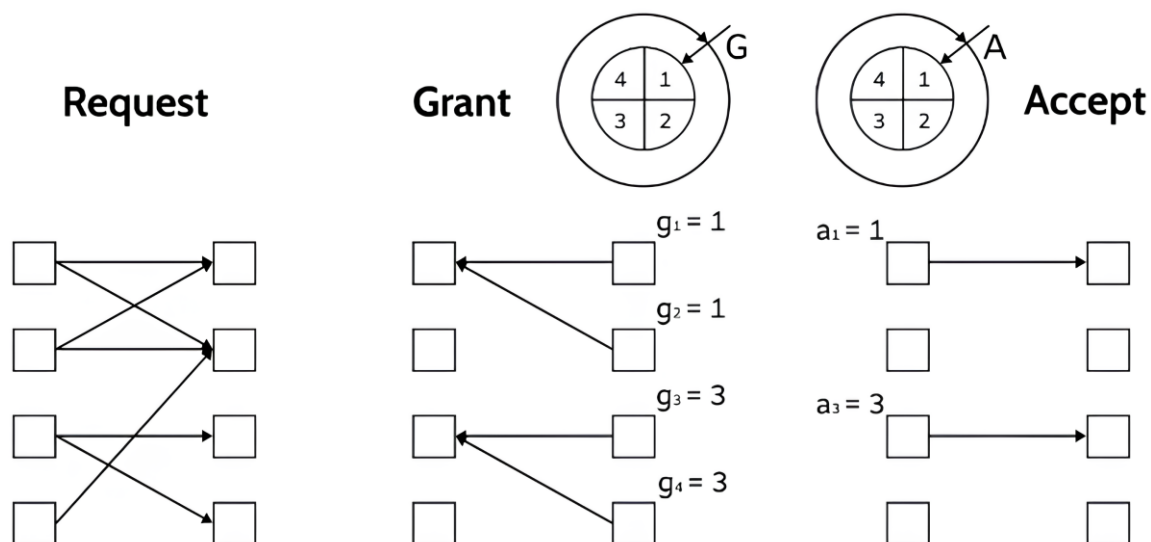


Figure 1. The grant–accept model [10,11]. Arrows indicate the direction of signal flow: requests are sent from inputs to outputs, grants are returned to inputs, and accepts confirm of the connection.

This idea was further developed in the IRRM (Iterative Round Robin Matching) [12]: instead of selecting requests randomly, a cyclic traversal (round-robin) of the grant and accept sets is used. In an improved version (iSLIP [11]) the output priority indicator (g_i) is updated only if the granted request was accepted by the input port. In the basic IRRM, this indicator is updated at each arbitration cycle regardless of whether the connection was successfully established.

Unlike iSLIP, in DRRM (Dual Round Robin Matching) [13], each input port can send only one request signal per cycle. This simplifies the hardware implementation of the arbiter and the connection establishment logic. CMCR (Centralized Multicast Contention Resolution) has greater functionality, as it can operate in multicast mode and consider different request priority levels.

PPE (Programmable Priority Encoder), a complex RR-based arbiter, was presented in [14]. It is an arbiter consisting of two FPE (Fixed Priority Encoder) blocks. The FPE serves its input requests according to a fixed priority order. Requests to the PPE are transmitted in parallel to the FPE blocks. The first FPE always has a fixed input priority order, and for the second FPE, a new highest priority pointer p is transmitted in each cycle (using RR). The second FPE block is the main one, but if it cannot provide grant to a request, the signal will be taken from the first FPE [15].

In 2DRR (2D round-robin) [16], the switch state is represented as a binary matrix RM , where $RM[i, j] = 1$ means that there is at least one packet for output port j at input port i . There are N predefined diagonals, and this set of diagonals is formed by shifting the main diagonal. A pattern sequence matrix (PSM) is used, which serves as a schedule for using the diagonals. In each time slot, the algorithm checks all diagonals but in a strictly defined order. If a cell in the request matrix is active ($RM[i, j] = 1$) and the corresponding input i and output j are not used, the corresponding connection is estab-

lished. The paper also presents Enhanced 2DRR algorithm. It uses N^2 variants of diagonals (sets of cells that do not intersect in rows and columns). This algorithm provides better fairness, but it introduces additional delays, and it is more difficult to implement.

In [17], the authors proposed an algorithm based on binary trees, specially designed for efficient hardware implementation. PRRA (Parallel Round-Robin Arbiter) is a complete binary tree (Figure 2). The leaves (*l*-nodes) store requests R_i and priority indicators H_j . The internal nodes (*i*-nodes) are combinational circuits that calculate the state of the subtree. The root (*r*-node) is responsible for the final granting decision. The arbiter searches for the first request ($R_i = 1$) using the round-robin principle, starting from the position marked by the pointer ($H_j = 1$). The route is encoded using two bits S^1S^0 in each node of the tree, which are calculated recursively from the bottom to the top level. In the improved version of this algorithm (IPRRA), modified *i*'-nodes generate G_L and G_R signals immediately after receiving S^1S^0 from the child nodes, without waiting for the G signal from the parent. GIPRRA (Grouped IPRRA) allows stages of the tree to be grouped to achieve a compromise between the gate delay, wire delay, and area.

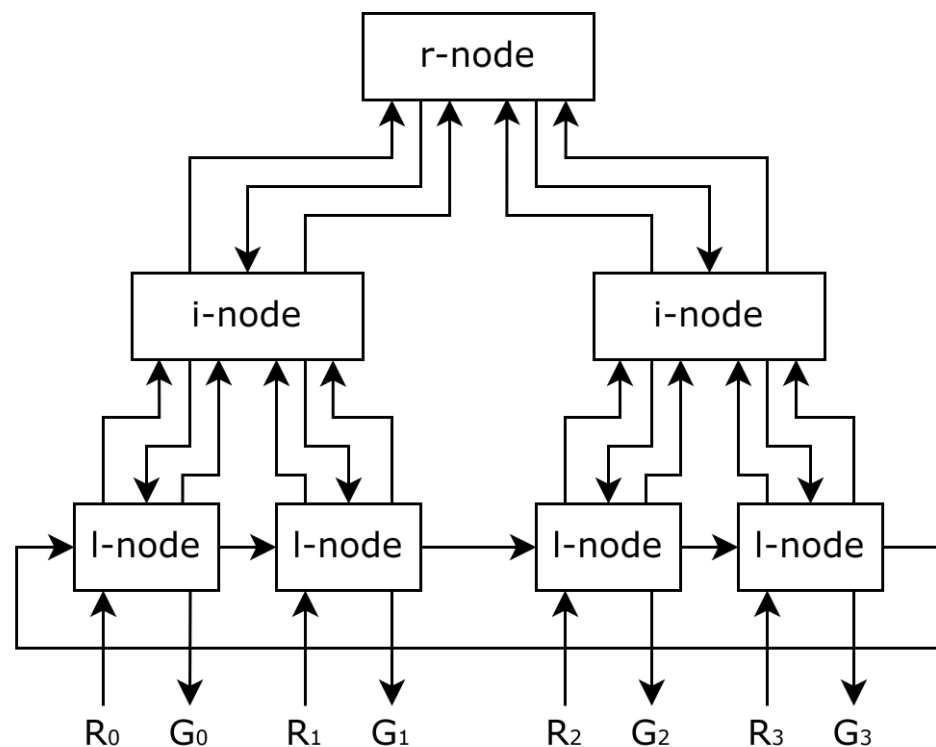


Figure 2. PRRA with 4 requests [17]. Arrows indicate the direction of signal propagation: request signals (R) travel upward from the inputs through the tree to the root, while grant signals (G) propagate downward back to the inputs.

A hybrid algorithm, combining the ideas of PPE and PRRA, known as HDRA (high-speed decentralized round-robin arbiter) [18] is organized as a DRR tree. At the *l*-node level, the AFRR (Absolutely Fair Round-Robin Arbitration Scheme) is used: it stores the input that won in the previous cycle and starts searching from the next input port in the current cycle. This ensures that there will be no starvation within the input group. Internal nodes and the root of the tree select the winner using a simple FPE block without dynamic priorities.

IRR (Index-based Round Robin) [4] operates with input port indexes instead of individual enable signals. A chain of multiplexers is used in which each multiplexer selects the request index with the highest priority in its range. The P (priority) input specifies which input port should be considered as the start of the priority cycle. After the grant

g_{id} is issued, its value is stored in the register. On the next clock cycle, the value of P is updated to $g_{id} + 1 \bmod N$, which provides a cyclic shift in priorities. If there are no requests, the arbiter retains the previous priority, which provides better fairness compared to PRRA.

In [19], the authors explored a scheme in which a single large RR arbiter is divided into smaller blocks to reduce the critical path and improve hardware efficiency. HRRA (Hierarchical Round Robin Arbiter) is divided into several layers of simple arbiters (Sub-RR, SRRA) that work in parallel and pass their decisions to the next levels. At the last level, there is one SRRA, which makes the final decision on granting connection permission. In [20], a hardware implementation of a fast modular arbiter was proposed: the input is a set of bits representing the values of requests R (0—no request from this node; 1—there is a request) and the priority value P in the form of a one-hot code (only one bit is equal to 1). The arbiter performs a round-robin search for the first active request, starting from the position with the highest priority.

Work [21] considers adaptive algorithms that combine RR and prioritization based on various parameters: rr-path, rr-dist, and rr-al. rr-path uses the distance between the source and the destination nodes (SDoffset). After locating the first request through RR, the algorithm continues monitoring remaining requests and may prioritize another with a higher SDoffset value to prioritize packets with a long path. rr-dist is similar to rr-path but uses the distance between the current and the destination nodes. This provides adaptivity during the routing process but rr-dist is less aggressive than rr-path. rr-al uses the difference between the number of packets received and sent in the port queues. The algorithm assigns higher priority to the most congested input ports.

RR-SFVP (Strong Fairness and Variable Priority) is an arbitration scheme designed to dynamically adjust request priorities based on packet aging, i.e., the waiting time accumulated by a request while seeking access to a shared network resource. The key idea of this approach is to progressively increase the priority of long-waiting requests, thereby preventing starvation and ensuring strong fairness among competing flows [22].

In [23], the MARX (Merged ARbiter and multipleXer) structure was presented, which combines an arbiter and a multiplexer into a single block, allowing arbitration and data transfer through the switch to be performed in a single clock cycle. MARX can operate in fixed priority mode (FPA) and in RR mode (the priority vector P is specified in the thermometric code).

2.2. Arbiters with Priority Based on Packet Characteristics

Age-Based Arbitration [24] uses packet age as a priority. Each packet receives an initial age equal to 0. A special timer increments at a specified rate. Each time a packet passes through a router, an offset is added to its age. This offset can vary for different network dimensions (X, Y, Z) and for different traffic classes (e.g., requests and responses).

The idea behind Probabilistic Arbitration [25] is to replace complex arbitration based on packet age with arbitration that assigns weight based on the hop count (distance) already accumulated by the packet. The arbiter can operate in three modes:

1. FW (Fixed Weight). The weight of the packet is calculated once it enters the network and considers the number of hops along the X-axis h_x and along the Y-axis h_y , as well as the degree of competition C (a static value depending on the location of the destination node in the topology).
2. CW (Constantly Increasing Weight). The weight of the packet starts at 1 and increases after each hop. When the packet reaches the destination node, its weight will be equal to the weight calculated in FW mode.

3. VW (Variably Increasing Weight). The weight of the packet also starts at 1, but in each router, it increases not by a fixed constant but by the current degree of competition for the output port. The degree of competition is equal to the number of packets that must be sent through this port.

2.3. Arbiters with Priority Based on Network Conditions

BA_RRA (Buffer Aware Round Robin Arbiter) [26] can operate in two modes:

1. High-priority mode is enabled when any input port buffer is full. In this mode, the arbiter considers requests only from ports with full buffers.
2. Normal mode is enabled when none of the buffers are full. In this mode, the arbiter works as a standard RR, sequentially selecting all active requests.

LQF (Longest Queue First) [27] takes into account the occupancy of router queues by assigning weights to them. Queues with higher load will receive a higher weight and are more likely to be served. However, this algorithm is too complex for hardware implementation as it uses a maximum-weight algorithm with a complexity of $O(N^3 \log(n))$. To solve this problem, an iterative version (iLQF) which works according to the grant-accept model was proposed. Each input sends requests to each required output, with its weight specified in the request. Each output selects the request with the highest weight and grants it permission to connect. In [28], it was also noted that LQF can lead to constant starvation of a non-empty queue; so, the OCF (Oldest Cell First) algorithm, which takes into account the waiting time of packets at the beginning of each queue and assigns weights to them, was proposed. Packets which have waited the longest are assigned a greater weight, and their probability of being served increases. To simplify the hardware implementation of the algorithm, an iterative version, iOCF, was also developed.

DAA (dynamically adaptive arbiter) [29] uses dynamic arbitration priority for each input port, which depends on the buffer status of that port. When an input port buffer becomes full, it is unable to receive further packets. For this reason, such ports are given high priority.

Works [30,31] describe dynamic arbiters based on a lottery mechanism. The D-L (Dynamic-Lottery) arbiter combines a lottery mechanism with real-time priority adaptation based on current port load. Each input port has a counter that represents its current weight or number of tickets. When a packet arrives at a port, the counter for that port is increased by 1; when a packet is successfully transmitted from a port, the counter for that port is decreased by 1. Thus, the number of tickets assigned to a port is directly proportional to the number of packets waiting in its buffer. LDPA (Load-based Dynamic Priority Arbiter) constantly updates information about the buffer occupancy of each input port. The average load of each port is calculated, and the ports are sorted in descending order. Based on the priority ratio for each port, an interval is created in the general ticket list. A random number is generated, and the port whose interval contains this number is granted access to the output port. LDPA implies a weighted and predictable distribution of priorities based on average load, while D-L is easier to implement and it effectively reduces buffer memory requirements under non-uniform traffic conditions.

2.4. Other Arbiters

Article [32] presented an arbiter based on two techniques:

- ATS (Arbiter Timing Speculation): The principle of dividing a long critical path of the arbiter (including all virtual channels) into several cycles to increase the clock frequency. Most simple arbiters complete their operation within a single clock cycle. If the request is not processed within s steps in one clock cycle, the arbiter state (priority pointer position) is saved, and the search continues in the next clock cycle.

- GPB (Group Priority Bypass): Reduction in arbitration delay in scenarios requiring more than one clock cycle (i.e., when the request is outside the first s steps) by skipping empty groups of virtual channels. The design incorporates a bus-like structure that allows the priority pointer to be passed from the current group not only to the adjacent group but also to groups spaced one, two, or more positions apart.

Parallel Pseudo-Round-Robin (P2R2) [33] is also designed to reduce the critical path during arbitration. P2R2 divides incoming requests into several groups and performs arbitration in all the groups in parallel. To select the winner among the groups, priorities between the groups are dynamically determined.

TCA (Thermometer-coded Arbiter) [34] is an optimized version of the PPE arbiter. PPE consists of two FPEs connected to a multiplexer, while TCA replaces FPE modules with thermometric encoders. FPE consists of a thermometric encoder and masking AND layer. In TCA, the masking layer is moved after the multiplexer and functions as an edge detector.

PPA (ping-pong arbitration) [35] is used to connect one input to one output. Inputs are grouped into pairs; each pair is controlled by a simple 2-input arbiter. The arbiters are then grouped into pairs again until a single tree is formed. The arbiter always selects the request from the input that currently has priority and has an active request. If an input has been served, a priority flag is toggled so that priority is given to a different input in the next cycle. Requests propagate upward through the tree, while grants travel downward; the final decision, made at the root of the tree, is passed down to the inputs. SA (Switch Arbiter) has an architecture similar to PPA but is based on 4-bit arbiters instead of 2-bit arbiters, since a single 4-bit arbiter occupies less area than two 2-bit arbiters.

In [15], an improved version of the PPA arbiter (IPPA) was proposed. IPPA consists of fewer logic elements; therefore, it occupies less area and has less delays. PLA (ping-lock arbiter) generates an additional single-bit lock output signal (lo), which is transmitted to a higher-level arbiter (parent). If the parent grants access to any of the input ports while the lo signal is set for it, it will not change the priority flag in the next cycle. This allows PLA to eliminate the fairness issues inherent in PPA and IPPA.

Lottery arbiter [36] is a probabilistic mechanism that assigns priorities based on lottery tickets. Each input port is assigned a certain number of tickets according to its priority. The higher the priority, the more tickets. In each arbitration cycle, a random number is generated in the range from 1 to the total number of tickets L . For each active request, its threshold is calculated. If the winning ticket number is in the range $[1, threshold\ N]$, permission will be granted to port N ; if the number is in the range $[threshold\ N, threshold\ S]$, permission will be granted to port S .

In MUCS (Matrix Unit Cell Scheduler) [37], the switch state is represented as an $N \times N$ matrix A , where element a_{ij} indicates the presence (and sometimes the priority) of a request at input i destined for output j . The algorithm iteratively selects requests to grant transmission permission. At each step, for each unprocessed element of the matrix, a weight is calculated using the formula $w_{ij} = a_{ij}/(\text{number of requests in row } i) + a_{ij}/(\text{number of requests in column } j)$. Requests with the least competition are selected first.

Another matrix arbiter, operating according to the LRS (Least Recently Served) scheme, was described in [38]. For N inputs, an $N \times N$ matrix is used. Each element M_{ij} stores a priority bit: $M_{ij} = 1$ means that input i has priority over input j ; $M_{ij} = 0$ means that input j has priority over input i . After servicing request r_i , row i of the matrix is reset to 0; column j of the matrix is set to 1. This approach ensures that the most recently served request becomes the lowest priority, while the next one becomes the highest. This is functionally equivalent to RR arbitration but realized in matrix form.

There are also various hybrid arbitration schemes that combine the advantages of two or more different arbitration policies [39–41]. Such approaches aim to balance competing objectives, such as fairness, latency reduction, and throughput optimization, by integrating multiple decision criteria within a single arbitration mechanism. However, the increased flexibility of hybrid arbiters comes at the cost of higher logical and physical complexity, which may lead to greater hardware overhead, increased critical path delay, and more challenging implementation in terms of timing closure and scalability. As a result, despite their potential performance benefits, the practical applicability of such arbiters often depends on careful trade-offs between architectural complexity and achievable QoS improvements in NoC designs.

Currently, arbitration techniques with machine learning (ML) are also being actively developed [42]. These approaches employ lightweight machine learning models to enable adaptive arbitration decisions based on observed network conditions. The main objective of such methods is to improve responsiveness to dynamic traffic patterns by learning or approximating optimal scheduling policies at runtime, while maintaining low hardware overhead suitable for NoC implementations.

In addition, dedicated arbitration schemes are being designed for NoC-based deep neural networks (DNNs) and convolutional neural networks (CNNs) [43–45]. These systems are characterized by distinctive communication patterns, where many-to-one (e.g., aggregation of partial results) and many-to-many (e.g., feature map exchanges between layers or processing elements) traffic flows are prevalent. Traditional arbitration policies are often not optimized for such structured and highly parallel communication behavior. Therefore, specialized arbiters are being designed to accommodate the specific characteristics of such workloads.

2.5. Conclusions About Existing Arbiters

The final classification of NoC arbiters organized as a tree is shown in Figure 3.

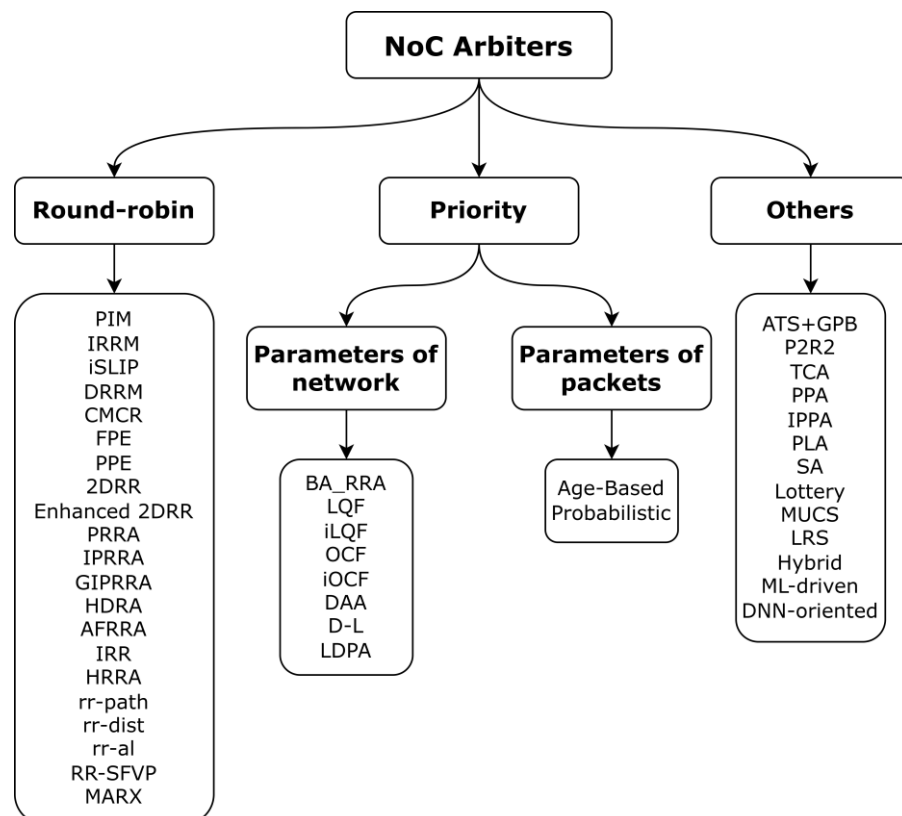


Figure 3. Classification of the NoC arbiters.

An analysis of known NoC arbiters shows that there are many different approaches to fair resource allocation and network performance improvement. Existing solutions cover all possible needs.

However, as the arbiter becomes more complex, overhead costs and implementation complexity increase, and in some cases, network performance may degrade. When using single-step and deterministic routing algorithms, complicating the arbiter does not significantly improve network performance; so, it is necessary to analyze simple lightweight arbiters.

The standard lightweight arbiter is round-robin, which is used in most modern solutions. However, it is necessary to explore new alternatives to round-robin as it is not universally applicable, and there are conditions under which other lightweight arbiters may be more suitable.

3. Materials and Methods

3.1. New Arbiters Under Consideration

Based on the above analysis, we decided to select arbiters that consider packet characteristics. This choice was made because packet characteristics are easy to implement and do not require additional changes to the network structure and routers. Another reason is that this class of arbiters remains insufficiently explored.

Section 2.2 discussed only two arbiters that consider packet characteristics: Age-Based Arbitration [24] and Probabilistic Arbitration [25]. They have two significant drawbacks: (1) the algorithms are quite complex, especially Age-Based Arbitration; (2) they give priority only to packets with the maximum values of calculated weights, i.e., only to packets that have been in the network the longest.

In this paper, we consider simplified and extended variations of the above arbiters. While existing approaches, such as Probabilistic Arbitration [25], rely on the distance from the source combined with complex weight calculations, we propose a lightweight alternative based solely on the hop count already traversed by the packet. In addition, we introduce a previously unexplored arbitration criterion—the remaining hop distance to the destination—and evaluate it in both priority directions. For each metric (hop count and remaining hop distance), we consider two variants, minimum-priority and maximum-priority, resulting in four arbitration policies:

1. Hopcount-min assigns higher priority to packets with the minimum hop count (packets that have traversed the fewest intermediate routers and are at an early stage of their route, typically closer to the source).
2. Hopcount-max assigns higher priority to packets with the maximum hop count (packets that have traversed the most intermediate routers and have resided in the network longer, being further along their routing path).
3. Distance-min assigns higher priority to packets with the smallest remaining hop distance to their destination (packets that are closest to completing their delivery).
4. Distance-max assigns higher priority to packets with the largest remaining hop distance to their destination (packets that are at an earlier stage of their end-to-end route and still require the greatest number of hops).

The choice of hop count and remaining hop distance as primary arbitration metrics is motivated by the following considerations. Unlike age-based or probabilistic schemes, which require per-packet timestamps, global counters, or complex weight calculations, the hop count between nodes is trivially computable in most NoC implementations. For hopcount-based arbiters, this eliminates the need for additional control logic, dedicated storage, or global synchronization. The remaining distance metric, while also simple, requires either computation at each hop or a precomputed distance table; we adopted the

latter approach, which trades a small amount of memory for low-latency arbitration. Both metrics are first-order indicators of packet progress and network occupancy: hop count reflects the service already consumed by a packet, while distance captures the remaining service required. Together, they directly capture the fundamental trade-off between urgency, aging, and progress that underlies QoS-aware arbitration. The exploration of both minimum and maximum variants of each metric yields four extreme arbitration policies, spanning the design space along two axes: packet maturity and spatial priority.

Hopcount-based arbiters require minimal modification of the network logic and flit structure: a single numeric field serving as a hop counter is stored in the packet header and incremented at each router. Distance-based arbiters were tested in two implementation variants: (1) distance calculation at each router, and (2) one-time precomputation of all pairwise distances stored in a table. The first option imposes a high computational load on each clock cycle and offers no advantage over other arbiters; therefore, the second implementation was used in all subsequent tests. In this configuration, distance-based arbiters do not perform runtime calculations but require additional memory to store the distance table.

3.2. Evaluation of Hardware Resources for the Implementation of Arbiters

Synthesis-based resource estimation of the proposed arbiters was performed using the Altera Quartus Prime synthesis tool (version 17.1; Intel Corporation, Santa Clara, CA, USA) targeting the Terasic DE1-SoC FPGA board (Terasic Inc., Hsinchu, Taiwan). The estimation provided the required numbers of logic elements and registers for implementing the arbiters in mesh topologies of sizes 4×4 , 8×8 , and 16×16 . Physical prototyping with on-board validation is planned as future work. The results are shown in Figure 4. Resources for the hopcount-max and hopcount-min arbiters, as well as for the distance-max and distance-min arbiters, are identical, since they differ only in the $+/ -$ signs for calculating the number of hops for the hopcount-max and hopcount-min arbiters, and for calculating the remaining distance for the distance-max and distance-min arbiters, respectively.

The ALM estimates account for the combinatorial logic of the arbitration policy itself: the comparators, the priority selection logic, and, for hopcount-based policies, the logic required to increment and evaluate the hop counter. The register counts correspond to the per-router flit storage and the internal state of the arbiter. Regarding the distance-based policies, the arbiter includes logic to compare the pre-computed remaining distances and select the winner. However, the $N \times N$ distance table itself is treated as a network-level configuration memory rather than per-router combinatorial logic. Its memory overhead depends heavily on the target technology and the network size. Consequently, this memory is not included in the per-router ALM/register counts. This separation ensures that the reported figures provide a fair and accurate comparison of the arbitration decision logic itself, which is the primary focus of this evaluation.

For the proposed arbiters, the number of registers required for operation corresponds to the flit size. For the round-robin arbiter, it is 3 bits larger than the flit size, which is due to the need to store the index of the selected channel. However, the increase in the flit size itself, and consequently the number of registers required to implement the hopcount-max and hopcount-min arbiters, grows faster than for other arbiters, as the number of registers for storing the hop count depends on the topology diameter and increases with diameter.

The least amount of logic resources (adaptive logic modules) is required to implement the round-robin arbiter, as the channel selection algorithm is reduced to only incrementing a single index. The hopcount-max and hopcount-min arbiters have a simpler channel selection algorithm than the distance-max and distance-min arbiters, but they

require modifying the field storing the hop count, which requires additional logic modules. Therefore, the distance-max and distance-min arbiters have an advantage for a 4×4 mesh topology, as they do not require modifying the data in the flit. For a 16×16 mesh, the hopcount-max and hopcount-min arbiters have a significant advantage. For an 8×8 mesh topology, the difference in the required number of adaptive logic modules for these arbiters is at the margin of error and differs only in the number of required registers. However, this difference is not significant.

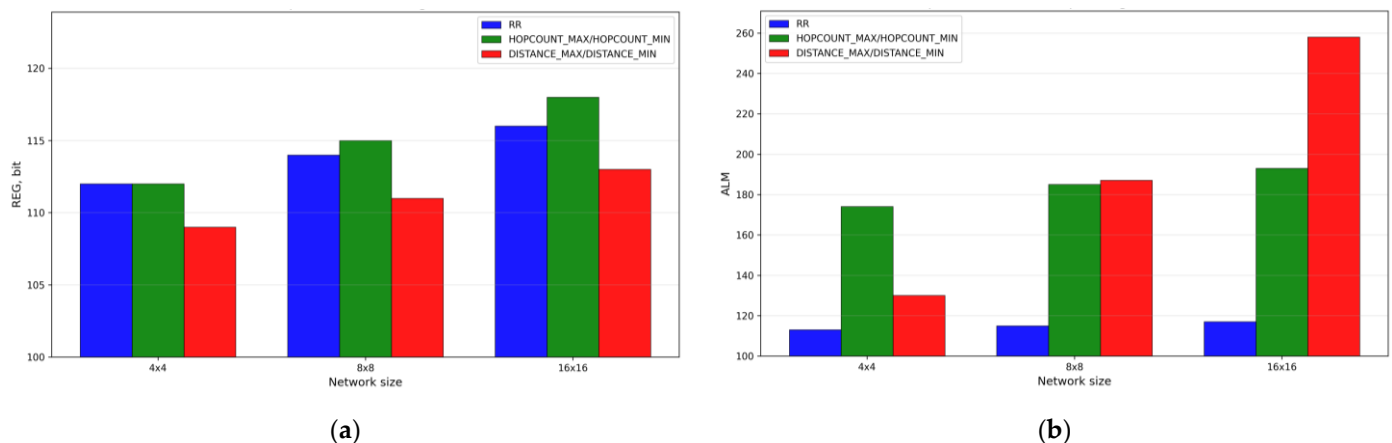


Figure 4. Required number of: (a) registers for hardware implementation of the proposed arbiters; (b) logic elements for hardware implementation of the proposed arbiters.

3.3. Implementation of the Arbiters into the NoC Simulator

The latest available open-source version of the Noxim [46,47] simulator was selected for modeling the NoC. In this context, packets are divided into the flow control units, called flits [48]. Flits can be classified as header, body, and tail [49]. The router in NoC reads the information contained in the header flit and makes routing decisions based on it. The remaining flit types are then forwarded automatically along the path established by the header flit [49]. In the original version of Noxim, the arbitration mechanism is implemented as a part of the flit transfer procedure in the router module and is not separated into a standalone configurable component. After the output direction reservation phase, a set of candidates is formed for each output port. If there is more than one candidate, the winner is selected using an equally probable random rule. Thus, the basic arbitration in Noxim corresponds to a non-deterministic policy of random conflict resolution.

The conceptual design of the new arbiters is structured as an extension of the existing selection strategy mechanism to create a full-fledged arbitration mechanism at the router's output ports. The implementation comprises three key layers: the strategy interface, the call point within the router, and the specific implementations of arbitration policies. At the interface level, a unified arbitration function is introduced, which accepts the router context and a list of candidates for a single output. The call occurs during the forwarding phase of the router: instead of a local random selection, the index of the winner is obtained using the active strategy, after which the selected flit is forwarded to the output port. This decomposition enables the use of a single router pipeline, requiring only the replacement of the candidate prioritization rule. Figure 5a shows a general scheme of the router and the location of the arbiter. The scheme for selecting an arbitration strategy in the Noxim simulator is shown in Figure 5b.

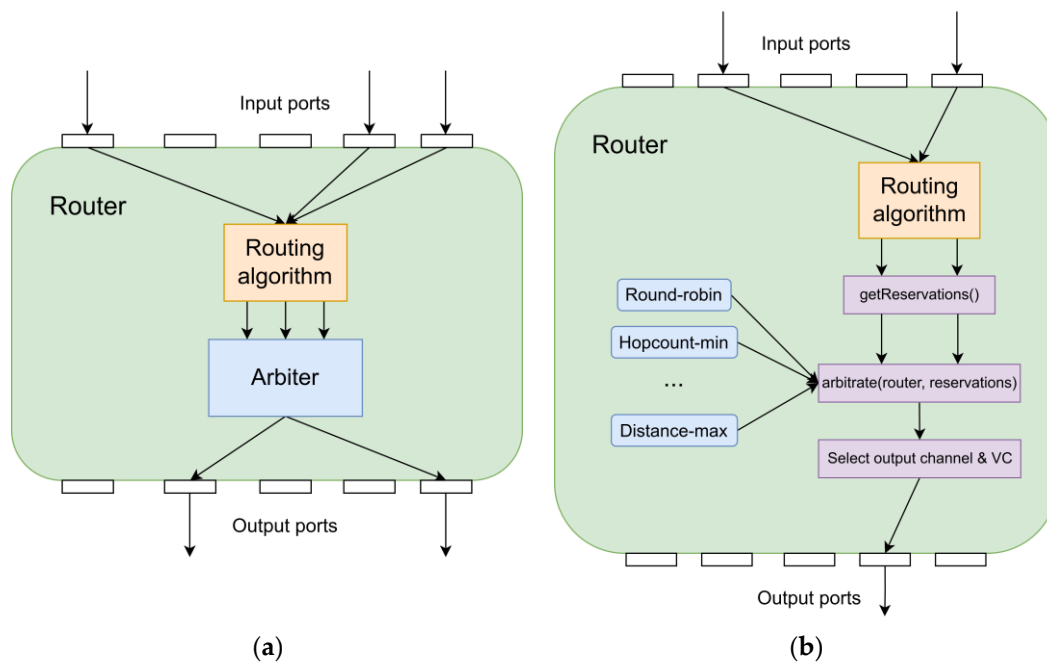


Figure 5. (a) General scheme of the router; (b) selection of the arbitration strategy in the Noxim simulator.

The default version of Noxim employs a random selection strategy when multiple input ports compete for the same output port. In our study, this built-in random arbitration is explicitly disabled and replaced by the considered deterministic policies. The active strategy is selected via the simulator's configuration parameter (*selection_strategy*), which allows one to switch between RR, hopcount-min, hopcount-max, distance-min, and distance-max without modifying the router pipeline. The random mechanism is retained only as a tie-breaker when all candidates have equal priority under the active policy; in all other cases, the winner is fully determined by the selected strategy.

To implement the round-robin arbiter in the Noxim simulator, a priority pointer was added to track the current selection position. The flit whose index matches the current priority, or, if none exists, the first flit encountered after it in the reservation list, is selected as the winner. After forwarding the selected flit, the priority is updated to the selected flit's index incremented by one.

To implement the arbiters that consider the number of hops, a *hop_no* counter was added to the header flit structure. This counter is initialized to zero when the flit enters the network and increments by 1 after each hop of the flit. The hopcount-min arbiter selects the flits with the lowest *hop_no* value, while the hopcount-max arbiter selects the flits with the highest *hop_no* value.

To implement the arbiters that consider remaining hop distance to the destination address, a two-stage scheme is used: preliminary distance estimation followed by winner selection. For mesh topologies, the distances are stored in a pairwise distance cache. Upon the first request or when the network size changes, an $N \times N$ matrix is formed, where N is the number of nodes in the NoC, and each element of the matrix represents the hop count between the corresponding nodes. After initialization, distance retrieval takes $O(1)$ time, which is critical for maintaining low arbitration latency.

During the arbitration phase, for each candidate from the set of reservations, a header flit is retrieved, and its residual distance is calculated. The distance-min arbiter selects the candidate with the minimum metric value, while the distance-max arbiter selects the one with the maximum.

3.4. Modeling Conditions and Parameters Considered

The following general parameters were defined for all the simulations:

- Topology: mesh;
- Router buffer size: 4 flits;
- Flit size: 32 bits;
- Packet size: 8 flits;
- Number of virtual channels: 1;
- Simulation time: 10,000 clock cycles;
- Network warm-up time: 1000 clock cycles.

Multiple virtual channels were not employed because their increasing number reduces the influence of the arbitration algorithm on overall network performance; i.e., arbitration is important in conditions of high competition for a single physical channel.

The performance of arbiters was studied under different network settings. The following parameters were varied during modeling:

- Network size (4×4 , 8×8);
- Packet injection rate (PIR) (from 0.01 to 0.1 in increments of 0.01);
- Traffic pattern (random, bit-reverse, butterfly, shuffle, transpose) [50];
- Routing algorithm (XY, west-first, north-last, negative-first, odd-even) [51].

A comprehensive simulation of NoC operation was conducted using all the parameters listed above and all the specified arbiters. The arbiters were compared based on the following parameters:

- Throughput;
- Average network delay;
- Maximum network delay.

Since the results of each simulation in Noxim may vary even under identical configurations, each experiment was repeated 10 times, and the reported measurements were averaged across all runs.

4. Results

Throughout this section, two types of figures are used. The win rate figures report the percentage of scenarios in which each arbiter achieves the best value of the considered metric. All other figures show the mean relative difference versus round-robin over 10 independent simulation seeds, with error bars indicating the corresponding sample standard deviation. In these relative performance figures, the black dashed horizontal line represents the baseline performance of the standard round-robin arbiter, and all other arbiters are compared as percentage improvements or degradations relative to this baseline. The small error bars confirm that the observed differences between arbiters are stable across runs rather than artifacts of simulation variability.

4.1. Average Results for All Tests

To determine the average results of all the tests, we checked how often each arbiter shows the best values for certain parameters. The preliminary summarized results are as follows: the best throughput is achieved when using standard RR followed by hopcount-max and distance-min arbiters (which are 17% less likely to be the best than RR). In terms of average delay, hopcount-max is the leader, distance-min is in second place, and RR is in the last place for this indicator. The best maximum delay is achieved when using hopcount-max and hopcount-min, and distance-max is in the last place for maximum delay.

Thus, on average across all simulations, the best performance is achieved when using the hopcount-max arbiter. Distance-min is also quite stable but inferior to the hopcount-max arbiter. Using RR allows for the best network throughput but worsens delays.

Details and analysis of the performance of different arbiters under different conditions are provided below.

4.2. Comparison by PIR Ranges

We study how varying network congestion levels affect the performance of arbiters. PIR defines the probability of packet generation at each network node in each clock cycle. We need to determine how different network loads affect the efficiency of the arbiters. PIR values greater than 0.1 correspond to network saturation, and the results become identical. We used PIR values from 0.01 to 0.1 in increments of 0.01. To determine the results, the PIR values were divided into three ranges:

1. Low PIR (0.01–0.03);
2. Medium PIR (0.04–0.07);
3. High PIR (0.08–0.1).

A low PIR corresponds to a network state without competition; at medium PIR values, competition for ports appears in the network, and throughput ceases to grow linearly; at high PIR, the network approaches saturation, and there occurs the maximum number of conflicts for ports.

To compare arbiters across different PIR ranges, we used a win rate criterion. A single experimental scenario is defined by the tuple (network size, routing algorithm, traffic pattern, PIR value, performance metric), where the metrics are throughput, average delay, and maximum delay. For each scenario, the metric value of every arbiter is the mean over 10 independent simulation runs at that configuration. An arbiter is declared a winner of the scenario if it achieves the best value of the considered metric (the highest throughput, or the lowest average or maximum delay). If several arbiters share the same best value, they are all counted as co-winners; consequently, the sum of win rates may exceed 100%.

Scenarios are then grouped into three PIR ranges. The win rate of an arbiter for a given metric and PIR range is the percentage of scenarios in that set for which the arbiter was a winner. For example, a 40% throughput win rate in a given PIR range means that the arbiter provided the highest throughput in 40% of the scenarios belonging to that range. This frequency-based criterion was chosen because absolute metric values vary strongly across traffic patterns, network sizes, and routing algorithms. The win rate therefore summarizes which arbiter is most often preferable over a wide range of operating conditions, rather than optimizing performance for a single configuration.

4.2.1. Results at Low PIR

Throughput and average delay at low PIR are highest for the hopcount-max arbiter (Figures 6 and 7). Hopcount-max is followed by distance-min, which is 10% worse. The lowest maximum delay is achieved with the RR arbiter (Figure 8). Thus, hopcount-max dominates in all metrics except maximum delay, indicating that some packets remain in the network for a prolonged time, which reduces fairness in resource allocation but substantially improves overall network performance.

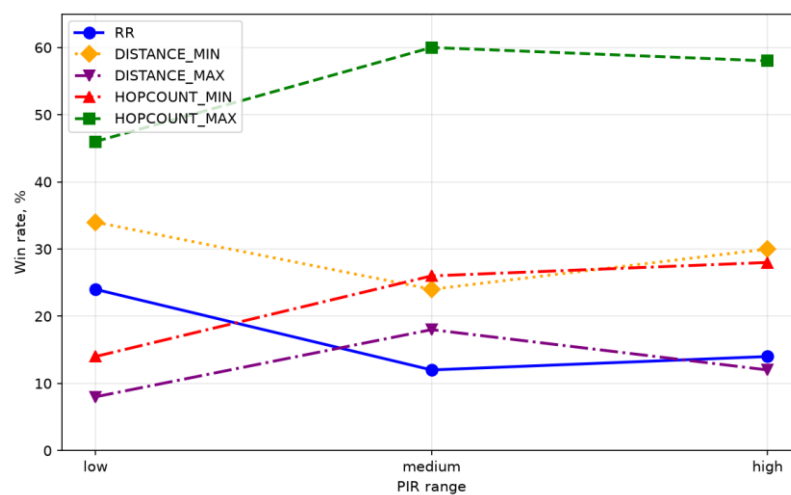


Figure 6. Comparison of the arbiters by throughput in different PIR ranges.

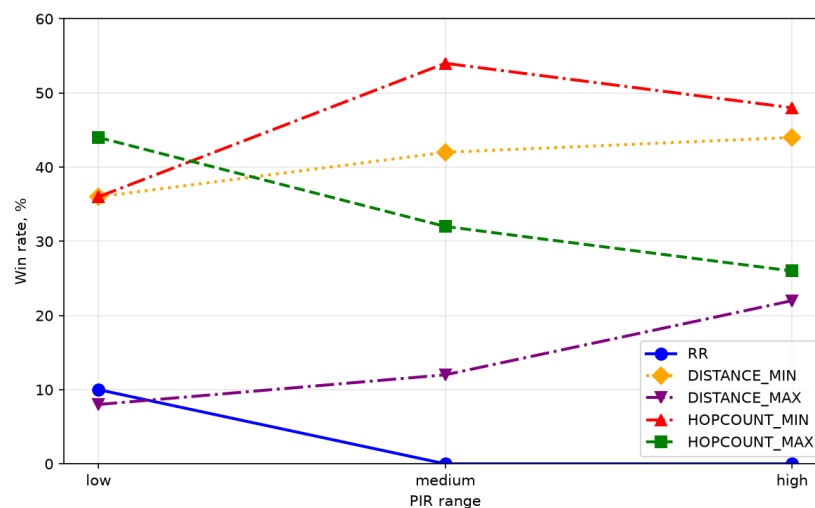


Figure 7. Comparison of the arbiters by average delay in different PIR ranges.

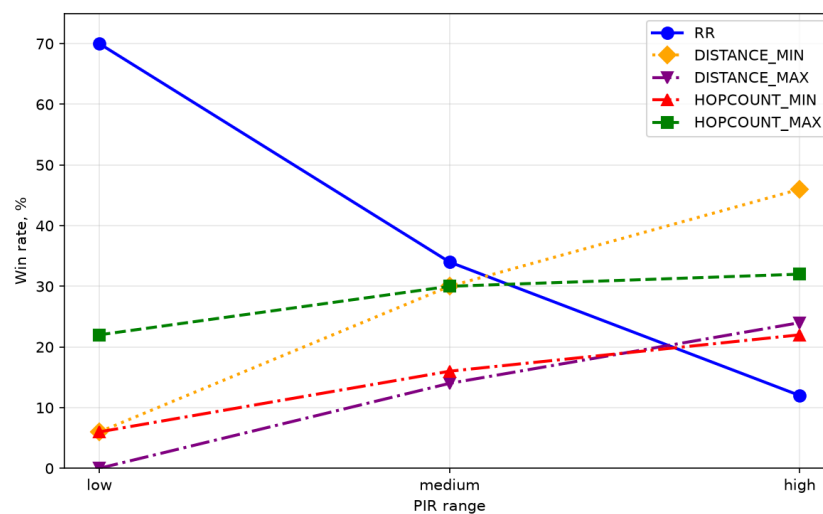


Figure 8. Comparison of the arbiters by maximum delay in different PIR ranges.

4.2.2. Results at Medium PIR

At medium PIR, the highest throughput is also achieved with hopcount-max (Figure 6), with a substantial margin of 35%. The lowest average delay is provided by hopcount-min, followed by distance-min with a 10% gap (Figure 7). The lowest maximum delay is again achieved with RR, although hopcount-max and distance-min are already approaching values of the RR, with a difference of 5% (Figure 8).

4.2.3. Results at High PIR

At high PIR, the highest throughput is also achieved with hopcount-max (Figure 6). The lowest average delay, as in the medium PIR case, is provided by hopcount-min and distance-min, with the difference between them becoming less than 5% (Figure 7). The lowest maximum delay is achieved with distance-min, while RR becomes the worst arbiter among all in this case (Figure 8).

4.2.4. Conclusions on the Impact of PIR

Overall, across all PIR ranges, hopcount-max dominates in throughput. In this case, average delay becomes a less informative metric, since a lower throughput implies that fewer packets are processed in the network, which decreases average delays. Maximum delay, by contrast, can serve as an indicator of fairness in resource allocation. At low PIR, the lowest maximum delay is achieved with RR, meaning that the other proposed arbiters allow large delays and stalls of certain packets when the network is lightly loaded. Thus, under light load, RR provides the most equitable resource distribution and prevents flow starvation. As PIR increases, the efficiency of RR in terms of maximum delay decreases, and at high PIR, RR becomes the least efficient arbiter among all. Under high load and high contention, simple round-robin no longer satisfies the maximum delay requirements, whereas priority-aware arbitration schemes distribute resources more efficiently and prevent excessively long stalls.

Owing to its substantial throughput advantage at any PIR, hopcount-max is recommended as the default arbiter. It also provides acceptable average delays and good maximum delays at medium and high PIR. If low maximum delay must be maintained under light load, the standard RR can be used. Distance-max can be considered the worst arbiter, as it provides nearly the lowest throughput and the highest delays, apart from RR.

In the subsequent results, some graphs are omitted for brevity. In such cases, this indicates that the missing results do not affect the overall ranking of the arbiters' performance, and only the most representative findings are presented.

4.3. Compatibility of Arbiters with Different Routing Algorithms

We investigate the impact of different routing algorithms on the efficiency of proposed arbiters.

In this subsection, each routing algorithm is evaluated under all traffic patterns listed in Section 3.4. For each routing algorithm, the arbiter that performs best on average over these traffic patterns and over the considered PIR range is reported below.

Table 1 shows the best arbiters according to various parameters in 4×4 and 8×8 mesh networks using different routing algorithms (XY [52], Negative-first [53], West-first [53], North-last [53], Odd-even [54]), as well as the average values of the corresponding parameters for each routing algorithm. For each combination of network size, routing algorithm, and metric, the reported average value is the global mean of the selected arbiter over all traffic patterns and packet injection rates. The best arbiter is chosen by the highest average relative gain against the other four arbiters. Relative gains are computed on traffic-averaged metric curves: at each PIR, the gain of a candidate arbiter versus each

of the other four is evaluated as a percentage improvement; these four gains are then averaged to obtain one value per PIR, and the average gain is the mean of those per-PIR values over the full PIR range. For the selected best arbiter, Table 1 also reports the maximum and minimum gains, i.e., the largest and smallest relative improvements observed over all PIR points and all pairwise comparisons with the other arbiters.

Table 1. The best arbiters for different routing algorithms.

Network Size	Routing Algorithm	Parameter	Best Arbiter	Average Value of Parameter	Average Gain, %	Max Gain, %	Min Gain, %
4 × 4	XY	Throughput (flit/cycle)	Hopcount-max	4.24 ± 0.02	2.31 ± 0.12	5.45 ± 0.21	−0.91 ± 0.30
		Average delay (cycles)	Hopcount-max	942.0 ± 17.6	12.1 ± 1.76	53.0 ± 1.26	−18.1 ± 11.0
		Maximum delay (cycles)	Distance-max	3995.2 ± 93.4	5.37 ± 2.73	57.9 ± 4.38	−73.0 ± 28.6
	Negative-first	Throughput (flit/cycle)	Hopcount-max	4.60 ± 0.02	6.14 ± 0.25	19.3 ± 1.10	−0.43 ± 0.27
		Average delay (cycles)	Hopcount-max	1059.8 ± 17.3	12.4 ± 1.58	66.8 ± 4.76	−23.9 ± 15.3
		Maximum delay (cycles)	Round-robin	4227.6 ± 45.1	8.36 ± 0.87	48.9 ± 4.43	−29.4 ± 3.76
	West-first	Throughput (flit/cycle)	Hopcount-max	4.61 ± 0.02	4.73 ± 0.19	13.2 ± 0.74	−0.48 ± 0.23
		Average delay (cycles)	Hopcount-max	1003.3 ± 14.0	13.3 ± 2.49	60.7 ± 5.56	−24.7 ± 30.4
		Maximum delay (cycles)	Round-robin	4833.3 ± 46.4	4.67 ± 3.40	49.5 ± 10.6	−48.9 ± 6.17
	North-last	Throughput (flit/cycle)	Hopcount-max	4.60 ± 0.02	6.19 ± 0.17	17.2 ± 0.82	−0.44 ± 0.24
		Average delay (cycles)	Hopcount-max	1045.7 ± 14.3	13.5 ± 1.42	70.1 ± 3.97	−17.4 ± 15.1
		Maximum delay (cycles)	Round-robin	4803.4 ± 42.5	6.65 ± 2.38	52.9 ± 7.91	−39.2 ± 4.00
	Odd-even	Throughput (flit/cycle)	Round-robin	4.57 ± 0.02	5.31 ± 0.24	17.7 ± 1.42	−0.91 ± 0.37
		Average delay (cycles)	Hopcount-max	1131.4 ± 17.2	11.4 ± 0.91	74.1 ± 2.87	−17.7 ± 1.45
		Maximum delay (cycles)	Round-robin	3855.7 ± 43.9	19.3 ± 1.90	61.4 ± 6.55	−21.5 ± 10.1
8 × 8	XY	Throughput (flit/cycle)	Hopcount-max	10.9 ± 0.02	8.10 ± 0.14	16.2 ± 0.72	−0.35 ± 0.35
		Average delay (cycles)	Hopcount-max	1185.5 ± 9.48	16.8 ± 0.95	60.7 ± 1.19	−9.08 ± 3.20
		Maximum delay (cycles)	Distance-max	6052.6 ± 86.6	8.66 ± 2.11	61.6 ± 3.37	−63.4 ± 38.8
	Negative-first	Throughput (flit/cycle)	Hopcount-max	10.9 ± 0.04	12.2 ± 0.40	28.9 ± 1.99	−0.43 ± 0.91
		Average delay (cycles)	Hopcount-min	1348.2 ± 14.0	17.0 ± 1.32	48.6 ± 3.19	−13.5 ± 9.06
		Maximum delay (cycles)	Distance-min	7110.9 ± 133.8	6.82 ± 1.71	31.5 ± 4.48	−30.9 ± 19.3
	West-first	Throughput (flit/cycle)	Hopcount-max	11.2 ± 0.06	11.4 ± 0.64	27.7 ± 1.33	−0.24 ± 0.37
		Average delay (cycles)	Hopcount-min	1440.4 ± 18.9	10.3 ± 1.52	40.8 ± 2.64	−37.1 ± 12.9
		Maximum delay (cycles)	Distance-min	7894.6 ± 115.4	2.62 ± 2.71	28.7 ± 11.6	−33.3 ± 21.5
	North-last	Throughput (flit/cycle)	Hopcount-max	11.4 ± 0.02	17.6 ± 0.32	37.8 ± 1.91	−0.07 ± 0.22
		Average delay (cycles)	Hopcount-min	1321.2 ± 22.6	9.09 ± 10.9	55.5 ± 3.41	−157 ± 112
		Maximum delay (cycles)	Distance-min	7890.2 ± 140.6	2.98 ± 2.24	34.6 ± 12.4	−24.2 ± 15.5
	Odd-even	Throughput (flit/cycle)	Hopcount-max	10.8 ± 0.05	12.8 ± 0.40	34.8 ± 1.69	−0.18 ± 0.15
		Average delay (cycles)	Hopcount-min	1513.9 ± 21.5	15.0 ± 1.31	42.5 ± 1.37	−12.1 ± 4.64
		Maximum delay (cycles)	Round-robin	7950.4 ± 25.3	4.27 ± 1.78	47.0 ± 14.1	−23.9 ± 4.70

Based on the table, in 4 × 4 mesh networks, XY provides the lowest average delay, odd–even provides the lowest maximum delay, and west-first provides the highest throughput (nearly matching negative-first and north-last). Hopcount-max and RR are the most suitable arbiters for almost all routing algorithms. Therefore, it is preferable to use west-first or odd–even routing with the RR arbiter under low network load and hopcount-max arbiter under medium and high loads.

In conjunction with the odd–even routing algorithm, RR is the most stable arbiter in terms of throughput (Figure 9). However, starting from PIR = 0.03, hopcount-max substantially outperforms RR in average delay (by 10–40%) (Figure 10). Since the difference in throughput does not exceed 1.5%, while the change in average delay is significant, hopcount-max is preferable on average. The worst arbiter in this case is hopcount-min, which is considerably inferior to all other arbiters across all metrics.

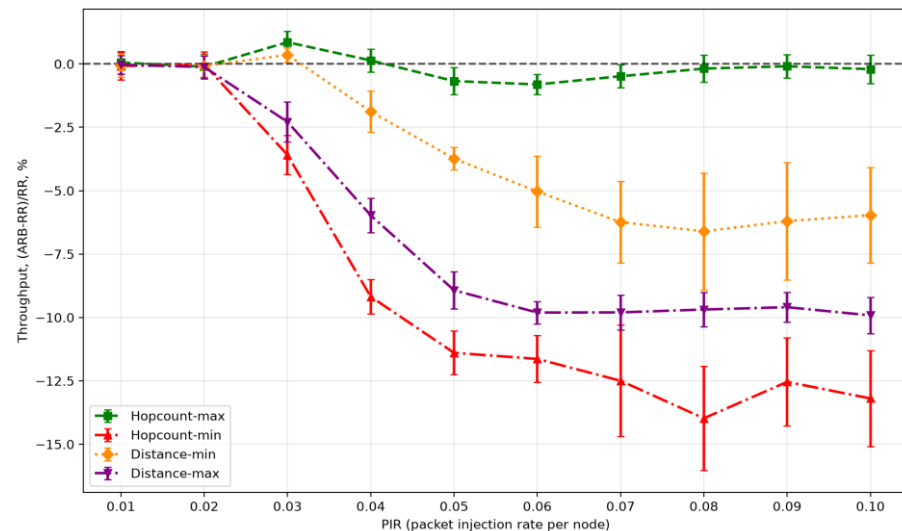


Figure 9. Comparison of the arbiters by throughput in a 4×4 mesh with odd-even routing.

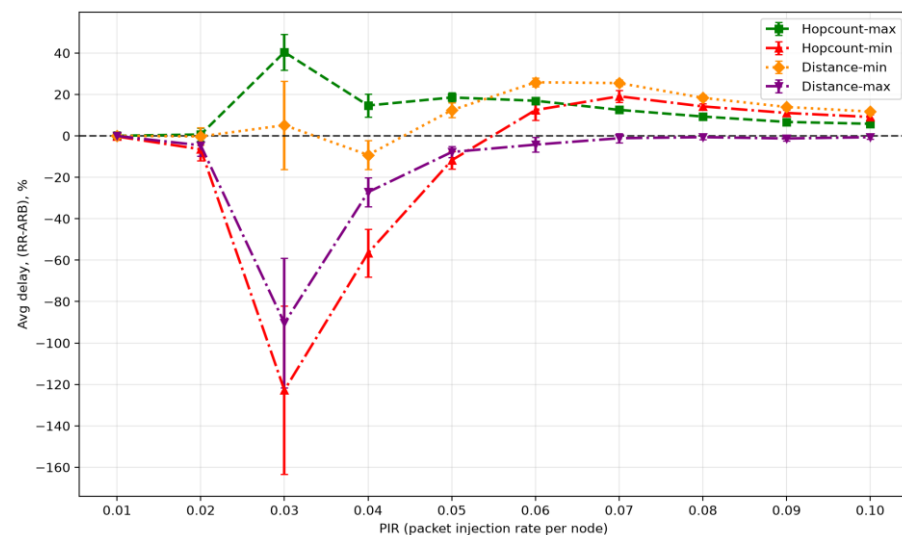


Figure 10. Comparison of the arbiters by average delay in a 4×4 mesh with odd-even routing.

The west-first routing algorithm is also most compatible with hopcount-max and RR (Figures 11 and 12). As PIR increases up to 0.03, hopcount-max begins to dominate in both average delay and throughput. The improvement in average delay with hopcount-max reaches 50%, while the improvement in throughput reaches 2%. The worst arbiter is again hopcount-min.

With odd-even routing, the performance of hopcount-max and RR is nearly identical at low PIR. However, with west-first routing under light load, hopcount-max is inferior to RR in average delay and exhibits a large standard deviation. Thus, hopcount-max is generally recommended under medium and high load, whereas RR is the most stable choice at low PIR.

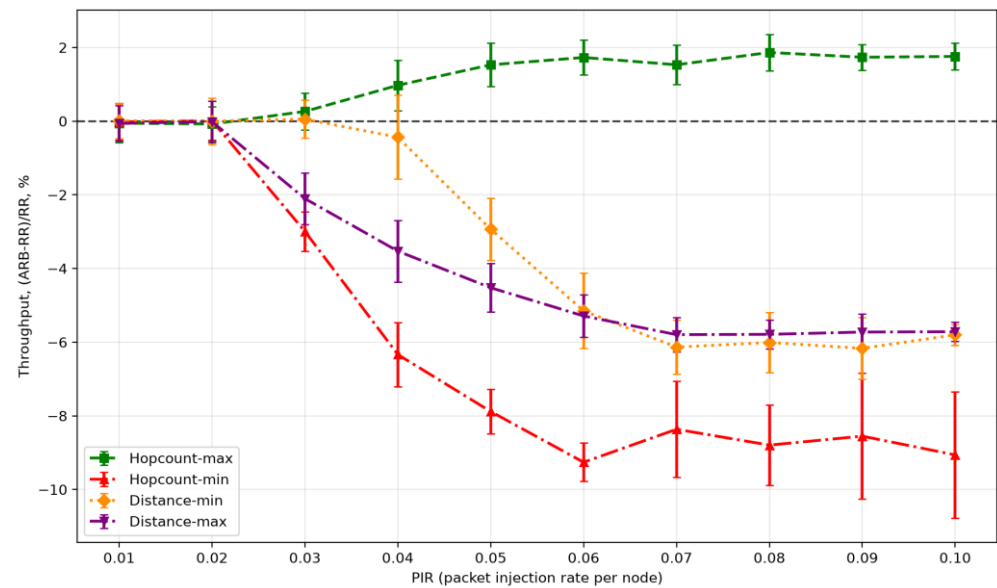


Figure 11. Comparison of the arbiters by throughput in a 4×4 mesh with west-first routing.

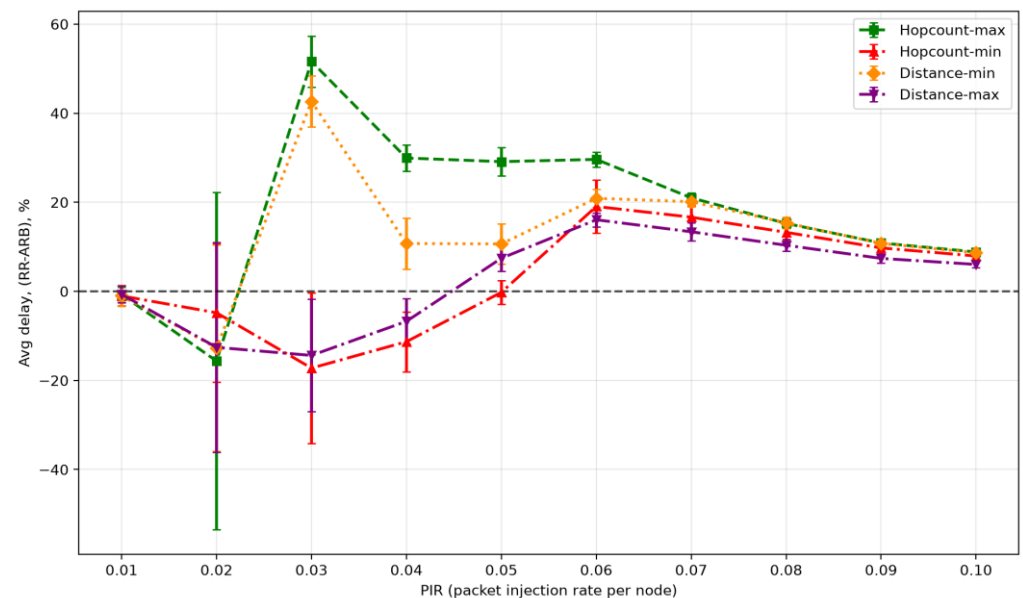


Figure 12. Comparison of the arbiters by average delay in a 4×4 mesh with west-first routing.

In 8×8 mesh networks, XY routing provides the lowest average and maximum delay. Throughput is the highest with north-last routing. For XY, the standard RR arbiter is not the most suitable: hopcount-max performs best for throughput and average delay, while distance-max is preferable for maximum delay. Figures 13 and 14 show a comparison of arbiters with XY routing in 8×8 mesh networks. Based on the figures, hopcount-max provides the highest throughput across the entire PIR range, whereas distance-max has an advantage in maximum delay only starting from PIR = 0.07, while being 13% inferior in throughput. Therefore, hopcount-max is preferable at any network load.

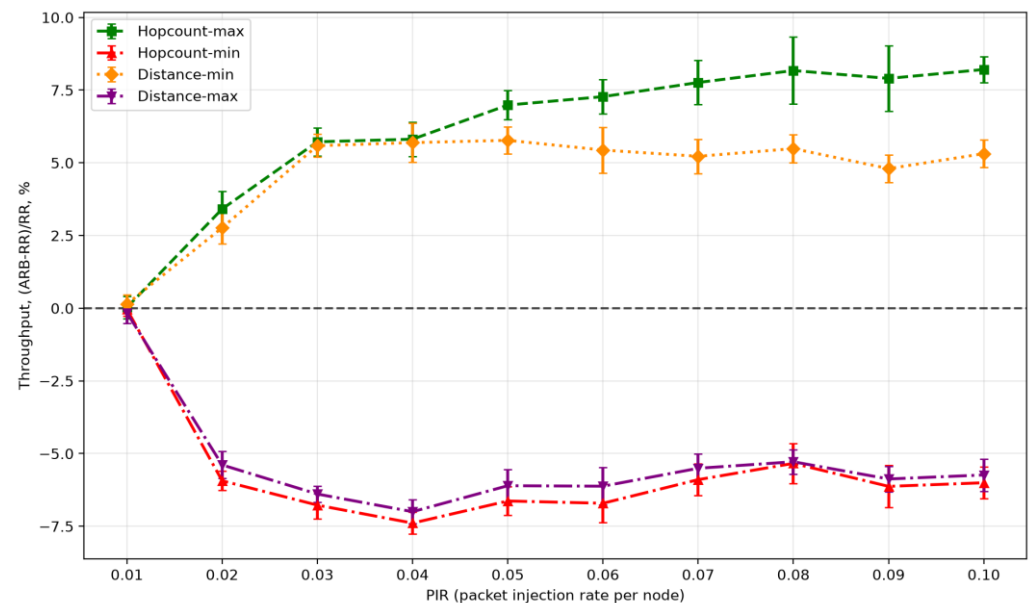


Figure 13. Comparison of the arbiters by throughput in an 8×8 mesh with XY routing.

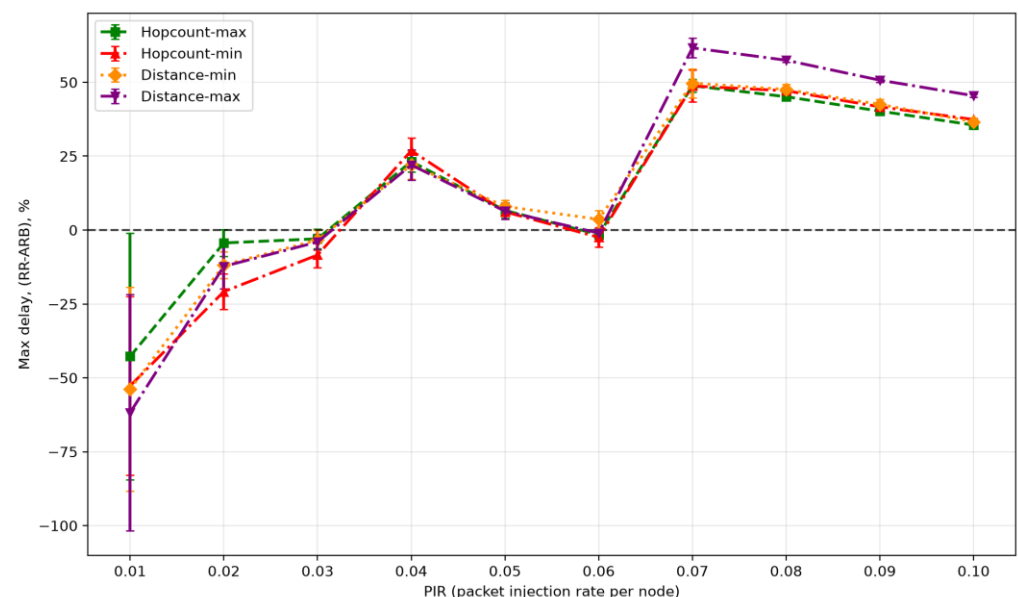


Figure 14. Comparison of the arbiters by maximum delay in an 8×8 mesh with XY routing.

With north-last routing in an 8×8 mesh, hopcount-max also outperforms all other arbiters in throughput across the entire PIR range (10% better than RR, and 25–30% better than hopcount-min and distance-max). RR is preferable in terms of maximum delay, since at injection rates below 0.03 the other arbiters exhibit maximum delays up to twice those of RR; however, under medium and high load, the other arbiters become slightly better than RR. In this case, hopcount-max is inferior in maximum delay by no more than 30% at PIR = 0.01.

Thus, hopcount-max is advantageous in almost all cases. In some configurations under light load, RR is effective, but under operating conditions with load and contention between packets, RR is inferior to hopcount-max. If hopcount-based arbiters cannot be implemented, distance-min is recommended, since it is usually the second most effective arbiter after hopcount-max. It is therefore beneficial to use arbiters that prioritize packets that have been in the network for a long time, i.e., to deliver packets that have nearly reached their destination faster.

4.4. Comparison of Arbiters Based on Traffic Pattern

We investigate the impact of different traffic patterns on the efficiency of proposed arbiters.

In this subsection, each traffic pattern is evaluated under all routing algorithms listed in Section 3.4. Figures below show the arbiter metrics for the indicated traffic pattern, with the values averaged over all routing algorithms.

Different traffic patterns have unique characteristics and regularities [50] that affect the distribution of results among arbiters. Below, we describe the regularities that arise for all traffic patterns under study.

4.4.1. Random Traffic Pattern

Figure 15 presents a throughput comparison for an 8×8 network; the arbiter distribution for a 4×4 network is similar. The highest throughput is provided by hopcount-max (with an improvement over RR of up to 30% in the 8×8 network and 9% in the 4×4 network). Hopcount-min is again the worst arbiter in terms of throughput, and distance-max is the second worst; both are more than 10% inferior to RR.

The average delay distribution differs between network sizes. For a 4×4 mesh, hopcount-max also provides the lowest average delay (with an advantage over RR up to 50%), while hopcount-min and distance-max perform worst (up to 140% and 120% worse than RR). When the network size increases to 8×8 , hopcount-min becomes the most stable arbiter in terms of average delay (Figure 16); however, given its considerable throughput disadvantage, this arbiter is not recommended. The second best in average delay is hopcount-max, which is also the throughput leader (Figure 15), and it is therefore the best choice for random traffic.

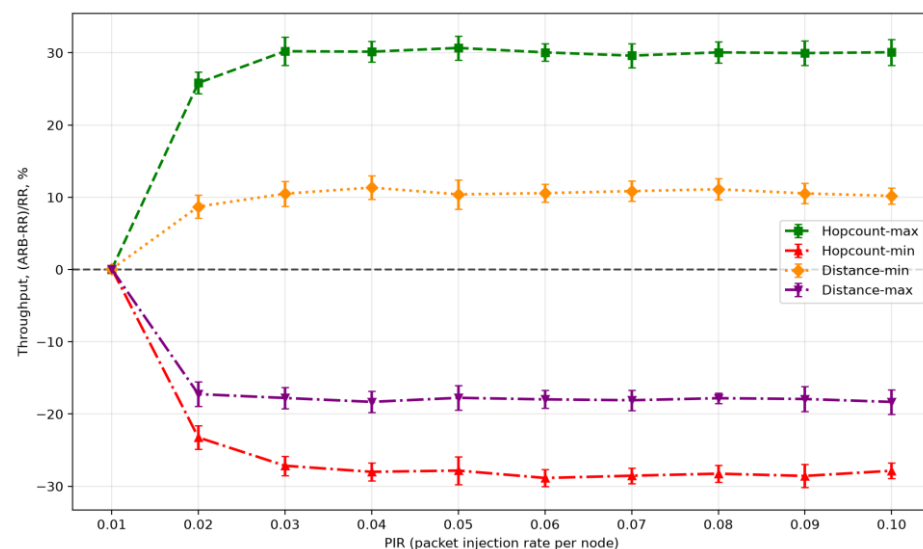


Figure 15. Comparison of the arbiters by throughput in an 8×8 mesh with random traffic pattern.

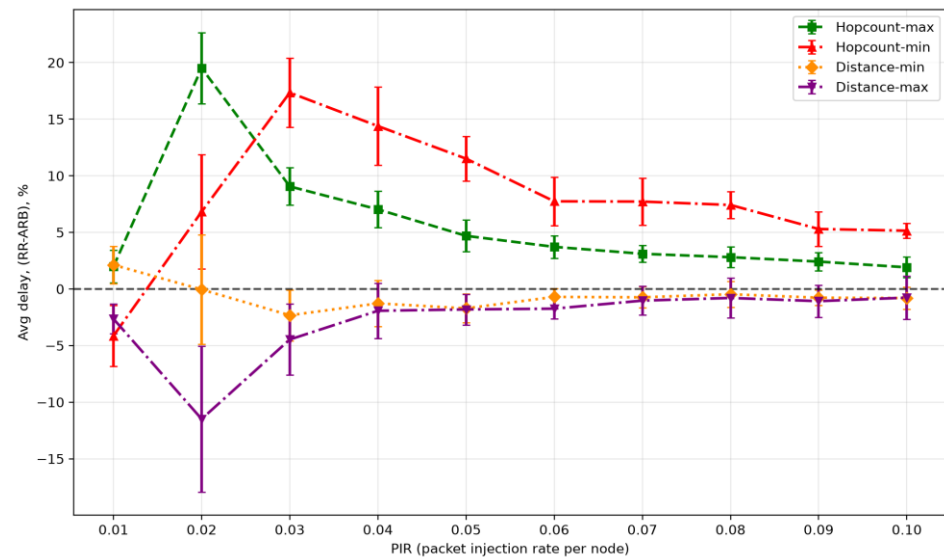


Figure 16. Comparison of the arbiters by average delay in an 8×8 mesh with random traffic pattern.

4.4.2. Bit-Reverse Traffic Pattern

In 4×4 networks with bit-reverse traffic, RR provides up to 10% higher throughput than distance-max, while hopcount-max provides up to 2% higher throughput than RR. Maximum delay clearly exhibits the trend in which RR is effective at low and medium PIR, whereas under high load, hopcount-max becomes the best arbiter (Figure 17).

When the number of cores in the network increases to 64, the maximum delay distribution remains almost the same (Figure 18), but new features emerge in throughput. At medium and high PIR, the values for RR and hopcount-max are almost identical, while under light load it becomes advantageous to use distance-min, since its use provides up to 4% higher throughput, while maintaining average delay 20–30% lower relative to RR (Figure 19). Thus, for bit-reverse traffic under light load, it may be beneficial to use the distance-min arbiter instead of hopcount-max or RR. This again confirms the benefit of faster service for the packets that have traversed most of their route.

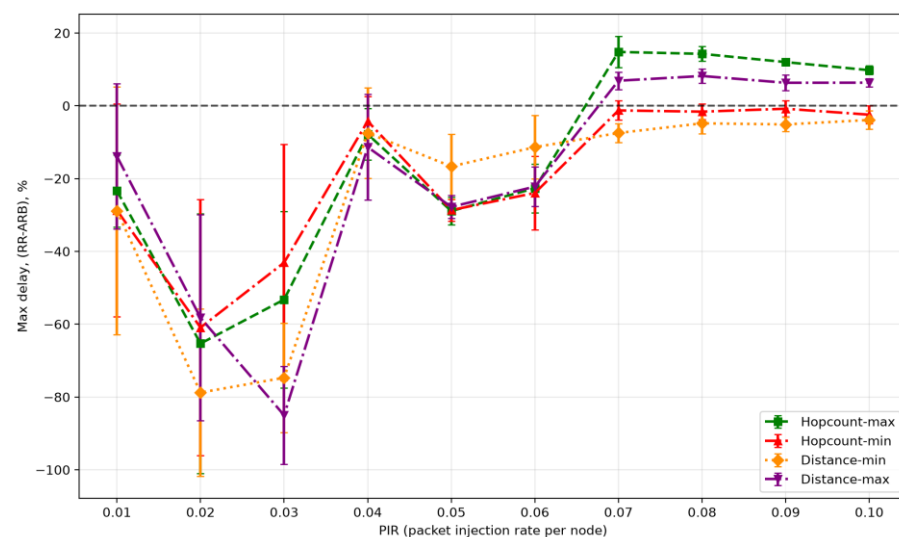


Figure 17. Comparison of the arbiters by maximum delay in a 4×4 mesh with bit-reverse traffic pattern.

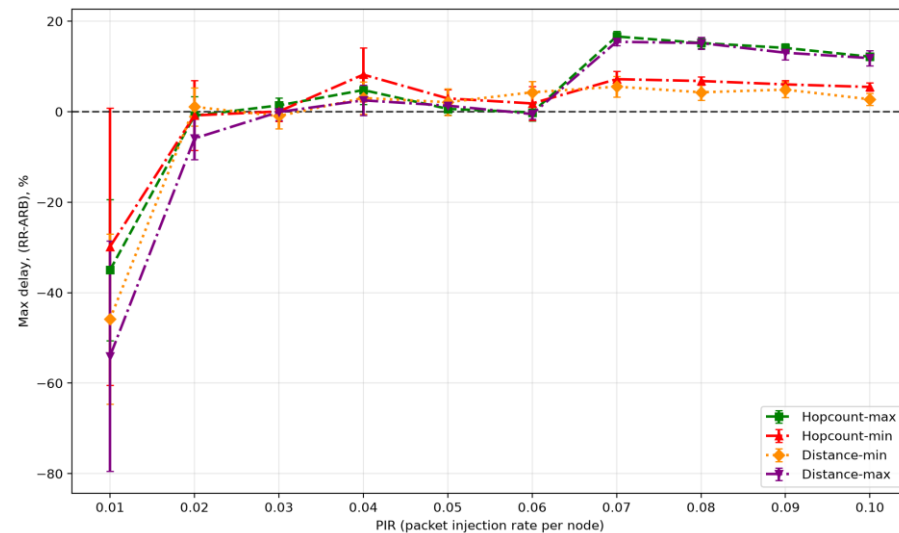


Figure 18. Comparison of the arbiters by maximum delay in an 8×8 mesh with bit-reverse traffic pattern.

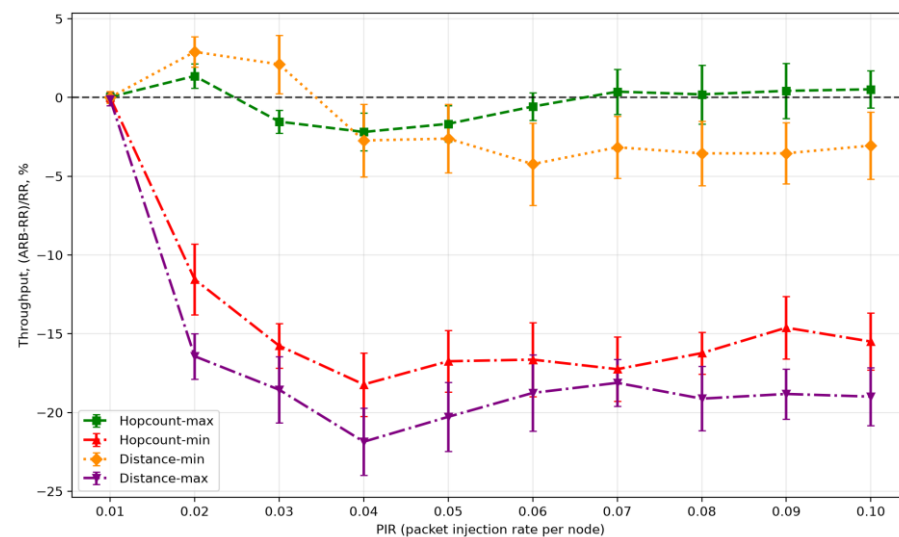


Figure 19. Comparison of the arbiters by throughput in an 8×8 mesh with bit-reverse traffic pattern.

4.4.3. Shuffle Traffic Pattern

Under shuffle traffic in a 4×4 network, all arbiters are inferior to hopcount-max in throughput (Figure 20). At high PIR, the lowest delays are achieved with distance-min (Figure 21); however, the substantial reduction in throughput (20% relative to RR) prevents it from being recommended for use in this case.

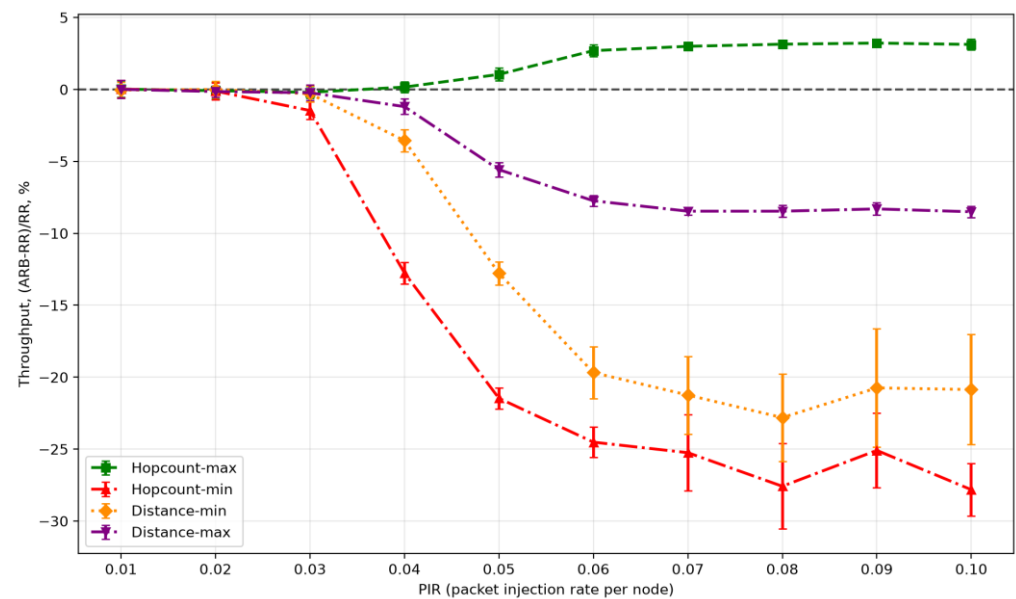


Figure 20. Comparison of the arbiters by throughput in a 4×4 mesh with shuffle traffic pattern.

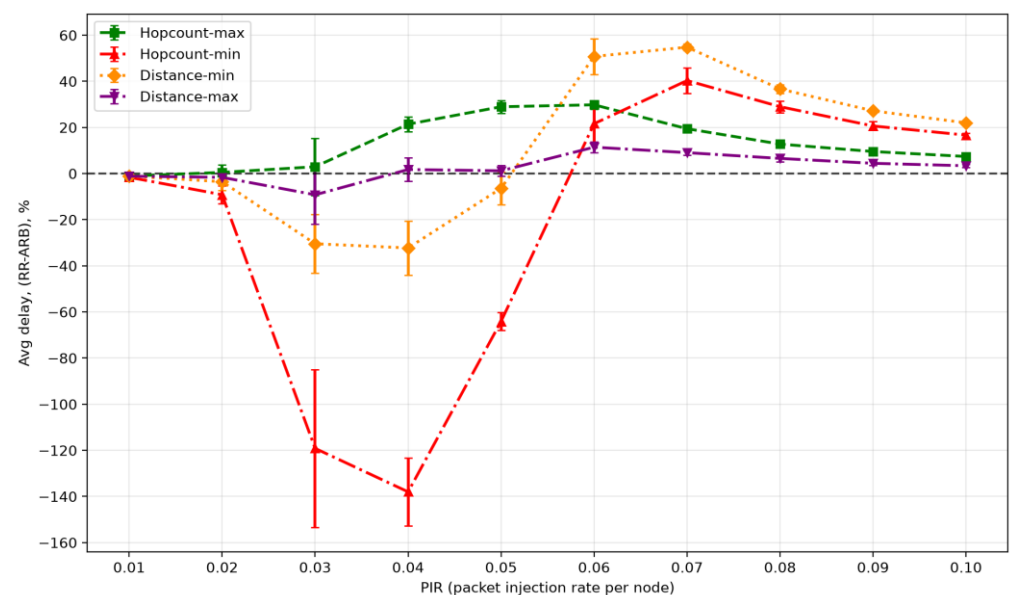


Figure 21. Comparison of the arbiters by average delay in a 4×4 mesh with shuffle traffic pattern.

In an 8×8 network, the throughput advantage of hopcount-max increases further, reaching 10–15% relative to RR (Figure 22). The lowest delay in this case is achieved with the hopcount-min arbiter; however, hopcount-min reduces throughput by 30–40%, which makes it less preferable overall.

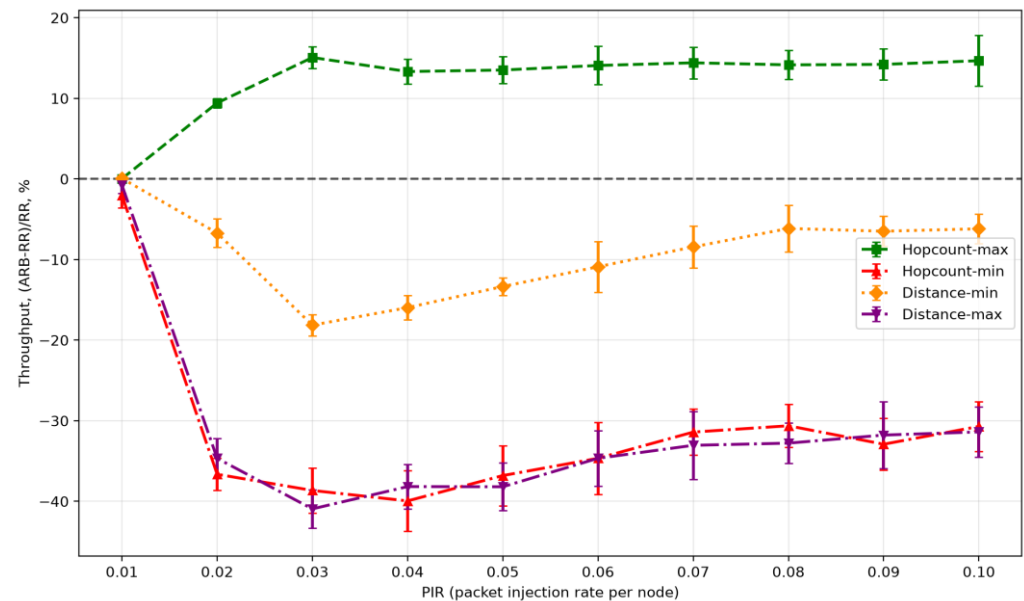


Figure 22. Comparison of the arbiters by throughput in an 8×8 mesh with shuffle traffic pattern.

4.4.4. Butterfly Traffic Pattern

Butterfly traffic differs substantially from all previously considered traffic patterns. For any network size, the results of hopcount-max are identical to those of distance-min, and the results of hopcount-min are identical to those of distance-max. Under butterfly traffic, all packets have the same route length L . Since $L = \text{hopcount} + \text{distance}$, a larger hop count implies a smaller remaining distance, and vice versa.

In a 4×4 network, the difference in throughput does not exceed 1%, and the standard deviation may exceed the difference between arbiters. Throughput varies randomly, and no leader can be identified by this metric. Average and maximum delays are similar for all arbiters except RR. The peak improvement in average delay relative to RR is reached in the medium PIR range (Figure 23), but the maximum delay degrades by 80% relative to RR (Figure 24). Thus, any arbiter except RR can be used under high load. Under medium load, the maximum delay requirements must be considered: if this parameter is critical, the standard RR should be used.

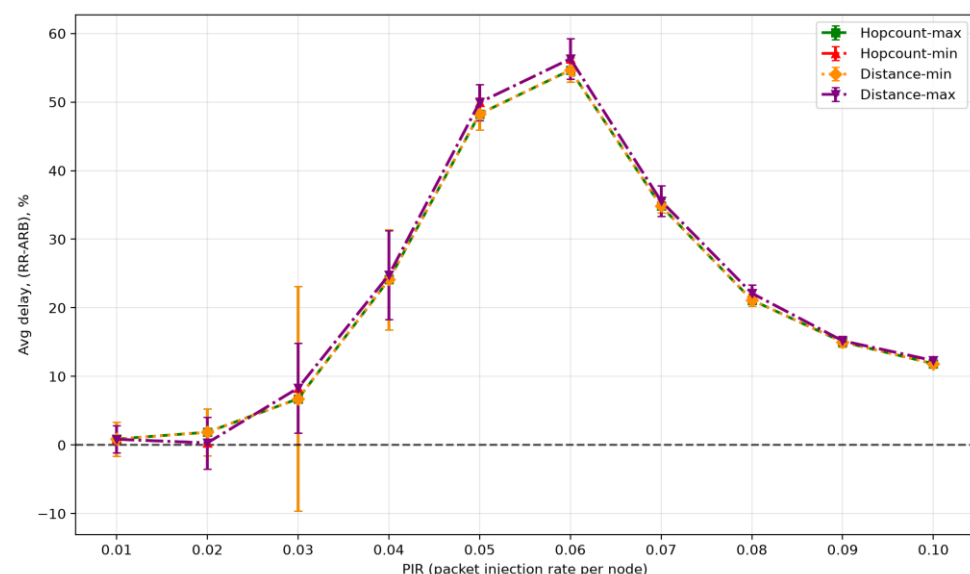


Figure 23. Comparison of the arbiters by average delay in a 4×4 mesh with butterfly traffic pattern.

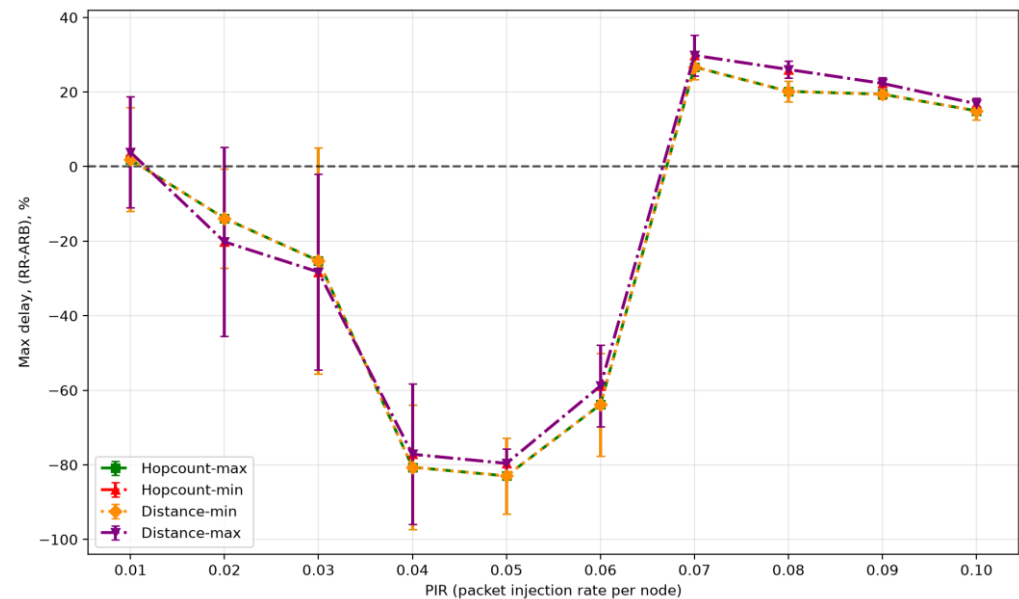


Figure 24. Comparison of the arbiters by maximum delay in a 4×4 mesh with butterfly traffic pattern.

As the network size increases to 8×8 , the dependencies become more stable, and the volatility of the results decreases. The average delay distribution remains approximately the same, but the maximum delay becomes acceptable at medium PIR and excessively high at low PIR (Figure 25). Hopcount-max and distance-min show a stable throughput improvement of 2–4% relative to RR (Figure 26), and both are therefore well suited for use in a network under medium and high load. Under light load, the maximum delay constraints again should be considered.

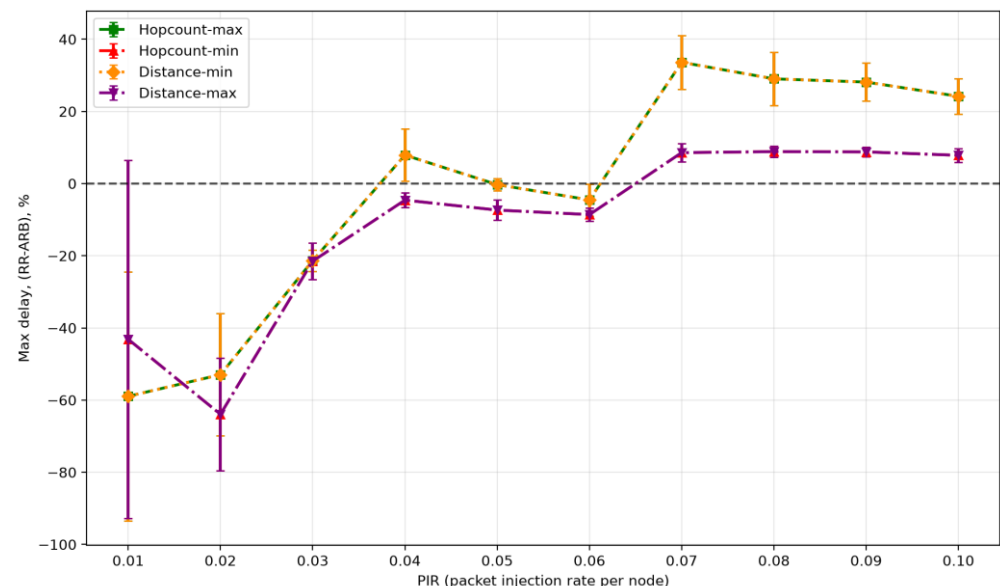


Figure 25. Comparison of the arbiters by maximum delay in an 8×8 mesh with butterfly traffic pattern.

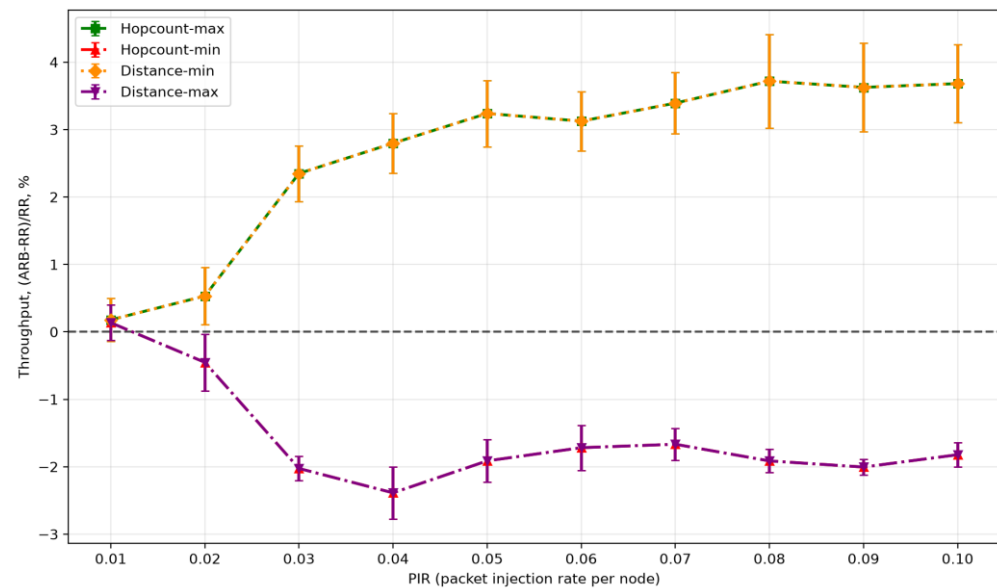


Figure 26. Comparison of the arbiters by throughput in an 8×8 mesh with butterfly traffic pattern.

4.4.5. Transpose Traffic Pattern

Transpose traffic also differs from the previously considered patterns, but the pairs of matching arbiters are reversed compared with butterfly traffic. For any network size, the results of hopcount-max are identical to those of distance-max, while those of hopcount-min are identical to those of distance-min. This follows from the geometric definition of transpose traffic: the route length is $L = 2|N - 1 - x - y|$. In contention points, packets with different route lengths meet. Compared with a packet originating from a shorter route, a packet on a longer route has traversed a greater number of hops and, simultaneously, has a greater number of hops remaining to its destination. Therefore, hopcount-max coincides with distance-max, and hopcount-min coincides with distance-min.

In a 4×4 network, the throughput difference is small: hopcount-min and distance-min differ from RR in throughput by 1% or less, whereas hopcount-max and distance-max underperform RR by 2–3% at medium and high PIR. The average delay of all alternatives to RR is noticeably better: the peak improvement occurs in the medium PIR range and reaches 65% (Figure 27). Maximum delay degrades significantly at low PIR but improves at high PIR (Figure 28). Thus, under high load, hopcount-min or distance-min is recommended, whereas under light load more stable performance is achieved with RR.

As the network size increases to 8×8 , the volatility of the results also decreases. Hopcount-min and distance-min provide a small but consistent throughput gain (1–2% relative to RR) (Figure 29) and retain a substantial improvement in average delay (50–60% relative to RR). Maximum delay at medium and high PIR is acceptable or better than RR, but at low PIR it degrades again (Figure 30). Hopcount-max and distance-max are inferior to RR in throughput (about 8% at medium and high PIR), although they remain better than RR in average delay. Therefore, for transpose traffic in an 8×8 mesh under medium and high load, hopcount-min and distance-min are the most appropriate; under light load, the choice of arbiter is again governed by maximum delay, and RR should be used if necessary.

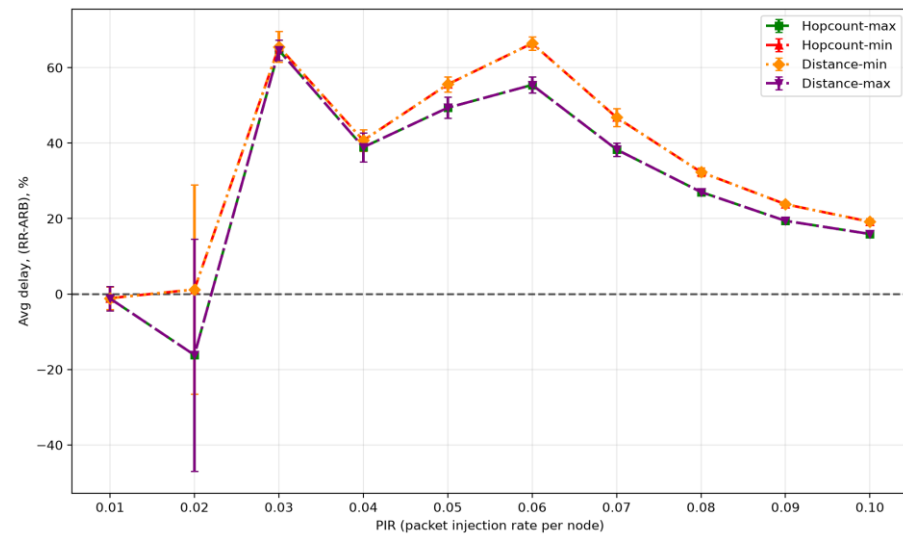


Figure 27. Comparison of the arbiters by average delay in a 4×4 mesh with transpose traffic pattern.

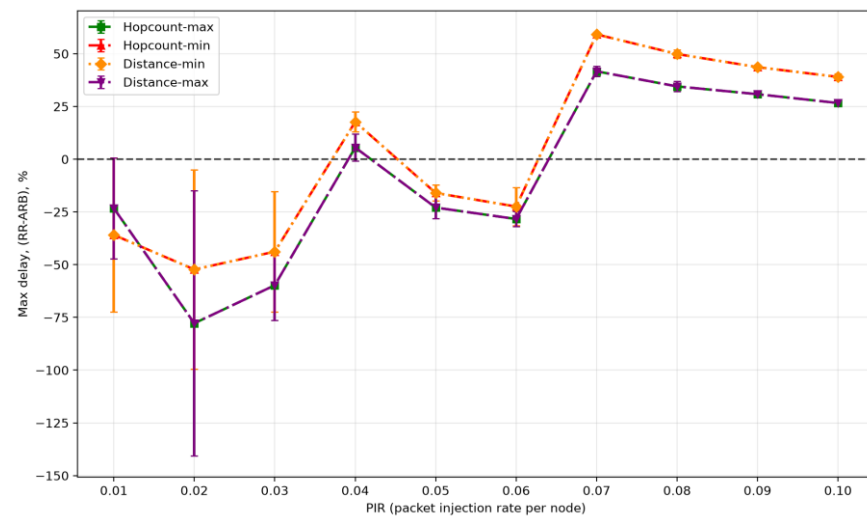


Figure 28. Comparison of the arbiters by maximum delay in a 4×4 mesh with transpose traffic pattern.

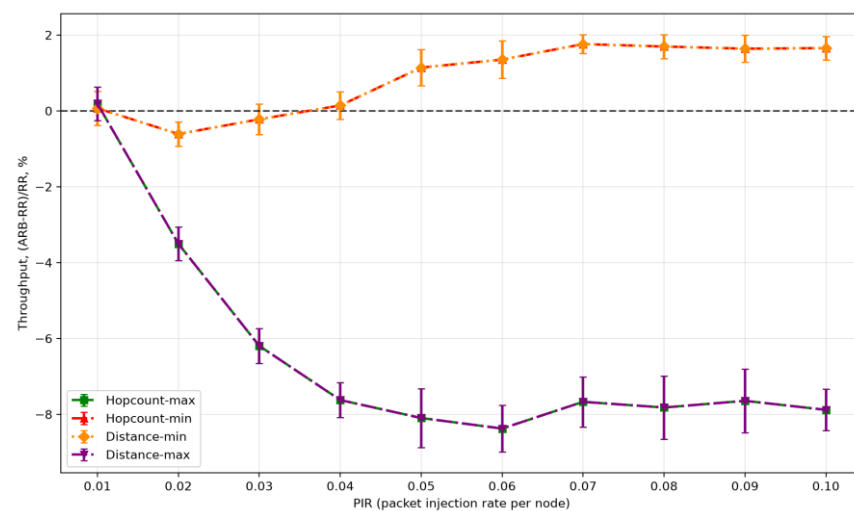


Figure 29. Comparison of the arbiters by throughput in an 8×8 mesh with transpose traffic pattern.

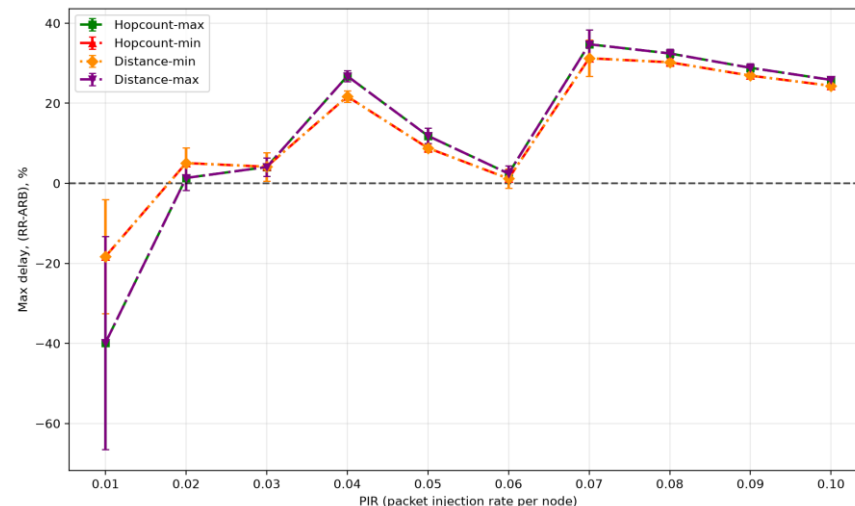


Figure 30. Comparison of the arbiters by maximum delay in an 8×8 mesh with transpose traffic pattern.

4.4.6. Conclusions on the Impact of Traffic Patterns

The traffic pattern strongly determines which arbiter is preferable. For random and butterfly traffic, the results largely confirm the earlier conclusions: hopcount-max and distance-min are the most effective alternatives to RR. Under random traffic, hopcount-max provides the clearest gains in both throughput and average delay, whereas distance-min is a weaker but still viable option; hopcount-min and distance-max are generally inferior to RR. Under butterfly traffic, special conditions arise: hopcount-max coincides with distance-min, and hopcount-min coincides with distance-max, since all packets have the same route length.

For shuffle and bit-reverse traffic, hopcount-max remains the best arbiter in terms of throughput and, in 4×4 networks, in terms of average delay. Hopcount-min and distance-max often reduce throughput and substantially worsen maximum delay, especially under low and medium load. RR remains preferable when maximum delay is critical. In the 8×8 shuffle case, distance-min becomes competitive in average and maximum delay, while hopcount-max leads in throughput. Transpose traffic differs from the other patterns: hopcount-min coincides with distance-min, and hopcount-max coincides with distance-max. The most effective pair is hopcount-min and distance-min, which provides better average delay and a small throughput gain over RR at medium and high load. Distance-max and hopcount-max are not among the best options. As with the other patterns, RR remains the safer choice for minimizing maximum delay under light load.

5. Discussion

When analyzing known NoC arbiters, various groups of arbiters and their characteristics were described. It was noted that the group of arbiters, which consider packet characteristics, is the smallest and least studied. There are many different arbiters that allow resources to be distributed in complex conditions, but such arbiters are poorly suited for small networks and simple routing algorithms as they require additional area and increase computation time; they are also difficult to design. Therefore, it was necessary to investigate lightweight arbiters capable of being quickly integrated into the NoC and used as an alternative to the standard round-robin, which does not always manage traffic effectively.

Several limitations define the scope of this study and should be considered when interpreting the results. The evaluation is limited to mesh topologies; other topologies

may exhibit different contention patterns and, consequently, different arbiter behavior. Performance simulations were performed for networks with 4×4 and 8×8 nodes, while the hardware resource estimates extend to 16×16 ; larger networks may amplify the differences between arbiters, especially for distance-based policies whose memory requirements grow with the number of nodes. Only synthetic traffic patterns were used; while these patterns are widely accepted as representative of a broad range of NoC workloads and congestion levels, they cannot capture all characteristics of real applications, and realistic traffic traces remain an important direction for future work. A single virtual channel per port was used deliberately: multiple virtual channels reduce contention for a single physical channel and thus weaken the influence of the arbitration policy, whereas the goal of this study was to isolate and compare arbitration policies under high competition; accordingly, the conclusions are most relevant for NoC designs with a single virtual channel per port. The study focuses on throughput and average/maximum delay; fairness, starvation probability, and per-flow latency distribution were not evaluated, and this remains an important direction for future work. Finally, the hardware evaluation was performed through FPGA synthesis rather than physical implementation: post-synthesis timing and power estimates were not extracted in this study, and the synthesis results should therefore be treated as an estimate of relative complexity rather than as final implementation figures. On-board validation and timing analysis remain future work.

The study revealed the strengths and weaknesses of the selected lightweight NoC arbiters. Across the full set of mesh sizes, routing algorithms, and traffic patterns, hopcount-max is the most frequent best choice, especially for throughput. Distance-min is often competitive and, under butterfly traffic, equivalent to hopcount-max; under transpose traffic, however, the effective pair is the opposite one (hopcount-min with distance-min). RR remains important when maximum delay is critical, particularly in the low PIR range, where priority-based arbiters can substantially increase the worst-case delay. Distance-max rarely provides the best results and should not be treated as a default recommendation, even for transpose traffic.

Hopcount-based arbitration is attractive not only because it often wins, but because it uses a metric that can be easily integrated into the flit and directly reflects how far the packet has already travelled. Remaining distance is a topological projection that ignores the actual route taken and the resources already consumed. At the same time, hopcount and distance are not interchangeable decision rules: their agreement depends on the traffic pattern, so both criteria must be interpreted in the context of the workload.

For implementation purposes, hopcount-max should be preferred over distance-min whenever their behavior is similar. Hopcount-max requires neither distance tables nor on-line distance computations, and the same counter can be reused to switch to hopcount-min, which is advantageous for delay under certain conditions. Distance-max, by contrast, rarely provides any benefit.

This work makes two main contributions. First, we introduce four lightweight arbitration policies based on hop count and remaining hop distance, including the previously unexplored remaining-distance criterion. Second, we provide an extensive assessment of these policies across a wide range of traffic patterns, routing algorithms, network sizes, and congestion levels. We show that hopcount-max is the most effective on average, and distance-min and hopcount-min are also effective in many scenarios. We also identify specific conditions where alternative policies are preferable, providing practical design guidelines for NoC architects.

Further improvements may be achieved through the design of hybrid or adaptive arbitration schemes. Such approaches involve the development of more complex arbiters that combine multiple priority criteria simultaneously or dynamically adjust their deci-

sion policy based on network conditions. In particular, adaptive arbiters may incorporate several arbitration rules within a single mechanism, or support runtime switching between different operating modes (e.g., max- and min-based prioritization strategies). This flexibility allows the arbitration logic to respond to varying traffic patterns and congestion levels, potentially improving overall network performance, fairness, and QoS guarantees.

At the same time, a comprehensive evaluation of QoS-related metrics, such as per-flow latency distribution, jitter, starvation probability, and fairness indices, remains an important direction for future work, as the present study has primarily focused on throughput and delay.

6. Conclusions

This paper presents a comparative analysis of lightweight arbiters for NoCs with mesh topology. A review of the scientific literature on NoC arbiters was conducted. Various classes of arbiters were identified, their operating logic was described, and their advantages and disadvantages were highlighted. Based on the analysis of the subject area, four lightweight arbitration policies were introduced and systematically evaluated under various NoC operating conditions.

The selected arbiters and standard round-robin approach were implemented in the cycle-accurate NoC simulator Noxim. The comparison was based on the three key parameters: throughput, average and maximum network delay. The dependencies of these parameters were studied when using the selected arbiters under various network operating conditions. Simulations were carried out with different network loads, routing algorithms, and traffic patterns. All comparisons were made on 4×4 and 8×8 networks.

As a result of the comparison, the most effective arbiters and combinations of arbiters were identified. Unique conditions (in which the best arbiters on average were inferior in performance to fewer effective alternatives) were noted. For the configurations considered, the most effective arbiters are those that give priority to packets that have been in the network the longest. Among the considered policies, the best arbiter is hopcount-max: an arbiter that gives priority to packets with the maximum number of hops from the source node.

An evaluation of the hardware resources required for the implementation of the proposed arbiters was carried out. It has been shown that the new arbiters are comparable in complexity to the classic round-robin arbiter and can be incorporated into practical NoC implementations.

The results of this work contribute to the ongoing research on QoS provisioning in on-chip communication systems, particularly in the context of lightweight arbitration for resource-constrained NoCs. The findings provide practical design guidelines for selecting appropriate arbitration policies depending on network load, traffic patterns, and performance objectives, thereby offering a valuable reference for NoC architects and system designers.

Future research will focus on several directions. Physical implementation of the proposed arbiters in FPGA prototypes will validate simulation results under real hardware conditions. Another direction is the design of a dynamic arbiter that switches between hopcount-min, hopcount-max, distance-min, and distance-max depending on real-time network state, potentially enhanced by lightweight machine learning or reinforcement learning agents. Evaluation in heterogeneous architectures, including chiplets and 3D NoCs, will further broaden the applicability of the proposed policies. A direct comparison with existing arbitration schemes from the literature, such as age-based and probabilistic arbitration, under the same experimental setup is also planned. Finally, an

automated design-space exploration framework integrating the proposed policies will support rapid trade-off analysis and industrial adoption.

Author Contributions: Conceptualization, M.M. and M.R.; data curation, M.M. and M.R.; formal analysis, M.M., R.S. and E.L.; funding acquisition, M.M., M.R. and A.R.; investigation, M.M.; methodology, M.M., R.S. and A.R.; project administration, A.R.; software, M.M. and E.L.; supervision, A.R.; validation, M.M., E.L., R.S. and M.R.; visualization, M.M.; writing—original draft, M.M. and E.L.; writing—review and editing, A.R. All authors have read and agreed to the published version of the manuscript.

Funding: The results of this research were obtained within the RSF grant No. 25-11-00248, <https://rscf.ru/en/project/25-11-00248/> (accessed on 5 September 2026).

Data Availability Statement: The data presented in this study are openly available in <https://github.com/mamalikov/Noxim-light-arbiters-LCirc-topology> (accessed on 5 September 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bjerregaard, T.; Mahadevan, S. A Survey of Research and Practices of Network-on-Chip. *ACM Comput. Surv.* **2006**, *38*, 1–51. <https://doi.org/10.1145/1132952.1132953>.
2. Grecu, C.; Pande, P.P.; Ivanov, A.; Saleh, R. Structured Interconnect Architecture. In *Proceedings of the 14th ACM Great Lakes Symposium on VLSI, Boston, MA, USA, 26–28 April 2004*; ACM: New York, NY, USA, 2004; pp. 192–195.
3. Benini, L.; De Micheli, G. Networks on Chips: A New SoC Paradigm. *Computer* **2002**, *35*, 70–78. <https://doi.org/10.1109/2.976921>.
4. Oveis-Gharan, M.; Khan, G.N. Index-Based Round-Robin Arbiter for NoC Routers. In *Proceedings of the 2015 IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, 8–10 July 2015*; IEEE: Piscataway, NJ, USA, 2015; pp. 62–67.
5. Wang, B.; Lu, Z. Flexible and Efficient QoS Provisioning in AXI4-Based Network-on-Chip Architecture. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2022**, *41*, 1523–1536. <https://doi.org/10.1109/TCAD.2021.3091410>.
6. Grot, B.; Keckler, S.W.; Mutlu, O. Preemptive Virtual Clock. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, 12–16 December 2009*; ACM: New York, NY, USA, 2009; pp. 268–279.
7. Goossens, K.; Dielissen, J.; van Meerbergen, J.; Poplavko, P.; Rădulescu, A.; Rijpkema, E.; Waterlander, E.; Wielage, P. Guaranteeing the Quality of Services in Networks on Chip. In *Networks on Chip*; Springer US: Boston, MA, USA, 2003; pp. 61–82.
8. Jain, K.; Singh, S.K.; Majumder, A.; Mondai, A.J. Problems Encountered in Various Arbitration Techniques Used in NOC Router: A Survey. In *Proceedings of the 2015 International Conference on Electronic Design, Computer Networks & Automated Verification (EDCAV), Shillong, India, 29–30 January 2015*; IEEE: Piscataway, NJ, USA, 2015; pp. 62–67.
9. Debbarma, J.; Banik, R.; Debbarma, D. Energy-Efficient Network-on-Chip Routers Using a Power Gated Round Robin Arbiter Ensuring Minimal Impact on Latency. *Results Eng.* **2025**, *28*, 108148. <https://doi.org/10.1016/j.rineng.2025.108148>.
10. Anderson, T.E.; Owicki, S.S.; Saxe, J.B.; Thacker, C.P. High-Speed Switch Scheduling for Local-Area Networks. *ACM Trans. Comput. Syst.* **1993**, *11*, 319–352. <https://doi.org/10.1145/161541.161736>.
11. McKeown, N. The ISLIP Scheduling Algorithm for Input-Queued Switches. *IEEE/ACM Trans. Netw.* **1999**, *7*, 188–201. <https://doi.org/10.1109/90.769767>.
12. McKeown, N.; Varaiya, P.; Walrand, J. Scheduling Cells in an Input-Queued Switch. *Electron. Lett.* **1993**, *29*, 2174–2175. <https://doi.org/10.1049/el:19931459>.
13. Chao, H.J.; Park, J.-S. Centralized Contention Resolution Schemes for a Large-Capacity Optical ATM Switch. In *Proceedings of the IEEE ATM Workshop Proceedings. “Meeting the Challenges of Deploying the Global Broadband Network Infrastructure” (Cat. No.98EX164), Fairfax, VA, USA, 26–29 May 1998*; IEEE: Piscataway, NJ, USA, 1998; pp. 11–16.
14. Gupta, P.; McKeown, N. Designing and Implementing a Fast Crossbar Scheduler. *IEEE Micro* **1999**, *19*, 20–28. <https://doi.org/10.1109/40.748793>.
15. Monemi, A.; Ooi, C.Y.; Palesi, M.; Marsono, M.N. Ping-Lock Round Robin Arbiter. *Microelectron. J.* **2017**, *63*, 81–93. <https://doi.org/10.1016/j.mejo.2017.03.004>.
16. LaMaire, R.O.; Serpanos, D.N. Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues. *IEEE/ACM Trans. Netw.* **1994**, *2*, 471–482. <https://doi.org/10.1109/90.336324>.

17. Zheng, S.Q.; Yang, M. Algorithm-Hardware Codesign of Fast Parallel Round-Robin Arbiters. *IEEE Trans. Parallel Distrib. Syst.* **2007**, *18*, 84–95. <https://doi.org/10.1109/TPDS.2007.253283>.
18. Lee, Y.-L.; Jou, J.M.; Chen, Y.-Y. A High-Speed and Decentralized Arbiter Design for NoC. In *Proceedings of the 2009 IEEE/ACS International Conference on Computer Systems and Applications, Rabat, Morocco, 10–13 May 2009*; IEEE: Piscataway, NJ, USA, 2009; pp. 350–353.
19. Kao, Y.-H.; Yang, M.; Artan, N.S.; Chao, H.J. CNoC: High-Radix Clos Network-on-Chip. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2011**, *30*, 1897–1910. <https://doi.org/10.1109/TCAD.2011.2164538>.
20. Dimitrakopoulos, G.; Chrysos, N.; Galanopoulos, K. Fast Arbiters for On-Chip Network Switches. In *Proceedings of the 2008 IEEE International Conference on Computer Design, Lake Tahoe, CA, USA, 12–15 October 2008*; IEEE: Piscataway, NJ, USA, 2008; pp. 664–670.
21. Zhu, X.; Zeng, H.; Huang, K.; Zhang, G. Round-Robin Based Scheduling Algorithms for FIFO IQ Switch. In *Proceedings of the 2008 IEEE International Conference on Networking, Sensing and Control, Sanya, China, 6–8 April 2008*; IEEE: Piscataway, NJ, USA, 2008; pp. 46–51.
22. Fard, E.S.; Jamali, J.; Masdari, M.; Majidzadeh, K. RR-SFVP: A Novel Arbitration Unit Design for NoC Router, Ingeniously Fusing the Round Robin Method with Strong Fairness and Variable Priority. *J. Electr. Comput. Eng. Innov.* **2025**, *13*, 283–298. <https://doi.org/10.22061/JECEI.2024.11230.779>.
23. Dimitrakopoulos, G.; Kalligeros, E.; Galanopoulos, K. Merged Switch Allocation and Traversal in Network-on-Chip Switches. *IEEE Trans. Comput.* **2013**, *62*, 2001–2012. <https://doi.org/10.1109/TC.2012.116>.
24. Abts, D.; Weisser, D. Age-Based Packet Arbitration in Large-Radix k-Ary n-Cubes. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, Reno, NV, USA, 10–16 November 2007*; ACM: New York, NY, USA, 2007; pp. 1–11.
25. Lee, M.M.; Kim, J.; Abts, D.; Marty, M.; Lee, J.W. Probabilistic Distance-Based Arbitration: Providing Equality of Service for Many-Core CMPs. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, Atlanta, GA, USA, 4–8 December 2010*; IEEE: Piscataway, NJ, USA, 2010; pp. 509–519.
26. Khan, A.A.; Mir, R.N.; Najeeb-ud-din. Buffer Aware Arbiter Design to Achieve Improved QoS for NoC. In *Proceedings of the TENCON 2017–2017 IEEE Region 10 Conference, Penang, Malaysia, 5–8 November 2017*; IEEE: Piscataway, NJ, USA, 2017; pp. 2494–2499.
27. McKeown, N.W. Scheduling Algorithms for Input-Queued Cell Switches. Ph.D. Thesis, University of California, Berkeley, CA, USA, 1995.
28. McKeown, N.; Mekkittikul, A.; Anantharam, V.; Walrand, J. Achieving 100% Throughput in an Input-Queued Switch. *IEEE Trans. Commun.* **1999**, *47*, 1260–1267. <https://doi.org/10.1109/26.780463>.
29. Liu, Y.; Jin, J.; Lai, Z. A Dynamic Adaptive Arbiter for Network-on-Chip. *Inf. MIDEM* **2013**, *43*, 111–118.
30. Wang, J.; Li, Y.; Peng, Q.; Tan, T. A Dynamic Priority Arbiter for Network-on-Chip. In *Proceedings of the 2009 IEEE International Symposium on Industrial Embedded Systems, Lausanne, Switzerland, 8–10 July 2009*; IEEE: Piscataway, NJ, USA, 2009; pp. 253–256.
31. Parvathi, S.; Umamaheswari, S. Load Based Dynamic Priority Arbiter for NoC Architecture. *J. Sci. Ind. Res.* **2022**, *81*, 495–504. <https://doi.org/10.56042/jsir.v81i05.49757>.
32. Zhao, B.; Zhang, Y.; Yang, J. A Speculative Arbiter Design to Enable High-Frequency Many-VC Router in NoCs. In *Proceedings of the 2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS), Tempe, AZ, USA, 21–24 April 2013*; IEEE: Piscataway, NJ, USA, 2013; pp. 1–8.
33. Bashizade, R.; Sarbazi-Azad, H. P2R2: Parallel Pseudo-Round-Robin Arbiter for High Performance NoCs. *Integration* **2015**, *50*, 173–182. <https://doi.org/10.1016/j.vlsi.2014.11.002>.
34. Fatih Ugurdag, H.; Baskirt, O. Fast Parallel Prefix Logic Circuits for N2n Round-Robin Arbitration. *Microelectron. J.* **2012**, *43*, 573–581. <https://doi.org/10.1016/j.mejo.2012.04.005>.
35. Chao, H.J.; Lam, C.H.; Guo, X. A Fast Arbitration Scheme for Terabit Packet Switches. In *Proceedings of the Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99. (Cat. No.99CH37042), Rio de Janeiro, Brazil, 5–9 December 1999*; IEEE: Piscataway, NJ, USA, 1999; Volume 2, pp. 1236–1243.
36. Wu, C.; Li, Y.; Chai, S.; Yang, Z. Lottery Router: A Customized Arbitral Priority NOC Router. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering, Wuhan, China, 12–14 December 2008*; IEEE: Piscataway, NJ, USA, 2008; pp. 411–414.
37. Duan, H.; Lockwood, J.W.; Kang, S.M. Matrix Unit Cell Scheduler (MUCS) for Input-Buffered ATM Switches. *IEEE Commun. Lett.* **1998**, *2*, 20–23. <https://doi.org/10.1109/4234.658616>.

38. Reehal, G.K. Designing Low Power and High Performance Network-on-Chip Communication Architectures for Nanometer SoCs. Doctoral Dissertation, The Ohio State University, Columbus, OH, USA, 2012.
39. Khan, A.A.; Mir, R.N.; Din, N.-U. Adaptive Hybrid Arbiter Design for Real-Time Traffic-Aware Scheduling. *Circuit World* **2022**, *48*, 185–203. <https://doi.org/10.1108/CW-10-2020-0268>.
40. Bedre, A.L.; Kumar, V.N. A Hybrid Arbiter to Accelerate Performance of High Speed Soc. In *Proceedings of the 2017 International Conference on Microelectronic Devices, Circuits and Systems (ICMDCS), Vellore, India, 10–12 August 2017*; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.
41. Fang, J.; Li, Y.; Wang, Y.; Jin, Z.; Yan, Y. Adaptive Arbitration Mechanisms for Heterogeneous NoCs under Diverse Load Scenarios. *J. Supercomput.* **2026**, *82*, 304.
42. Wang, S.; Zhang, X.; Dong, D.; Li, C.; Wang, Z.; Zhang, Z. LARE: A Linear Approximate Reinforcement Learning Based Adaptive Routing for Network-on-Chips. In *Proceedings of the 2023 IEEE International Symposium on Circuits and Systems (IS-CAS), Monterey, CA, USA, 21–25 May 2023*; IEEE: Piscataway, NJ, USA, 2023; pp. 1–5.
43. Fan, W.; Li, S.; Zhu, L.; Lu, Z.; Li, L.; Fu, Y. Communication Synchronization-Aware Arbitration Policy in NoC-Based DNN Accelerators. *IEEE Trans. Circuits Syst. II Express Briefs* **2024**, *71*, 4521–4525. <https://doi.org/10.1109/TCSII.2024.3395054>.
44. Dehghani, A. An Efficient NoC-Based Architecture for Convolutional Neural Networks on Multicore Systems. *J. Supercomput.* **2025**, *81*, 1066. <https://doi.org/10.1007/s11227-025-07526-x>.
45. Zhu, W.; Chen, Y.; Lu, Z. NoCDAS: A Cycle-Accurate NoC-Based Deep Neural Network Accelerator Simulator. *ACM Trans. Model. Comput. Simul.* **2025**, *35*, 1–24. <https://doi.org/10.1145/3729169>.
46. Catania, V.; Mineo, A.; Monteleone, S.; Palesi, M.; Patti, D. Noxim: An Open, Extensible and Cycle-Accurate Network on Chip Simulator. In *Proceedings of the 2015 IEEE 26th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Toronto, ON, Canada, 27–29 July 2015*; IEEE: Piscataway, NJ, USA, 2015; pp. 162–163.
47. Romanov, A.Y.; Myachin, N.M.; Lezhnev, E.V.; Ivannikov, A.D.; El-Mesady, A. Ring-Split: Deadlock-Free Routing Algorithm for Circulant Networks-on-Chip. *Micromachines* **2023**, *14*, 141. <https://doi.org/10.3390/mi14010141>.
48. Concer, N.; Petracca, M.; Carloni, L.P. Distributed Flit-Buffer Flow Control for Networks-on-Chip. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Atlanta, GA, USA, 19–24 October 2008*; ACM: New York, NY, USA, 2008; pp. 215–220.
49. Rad, F.; Reshadi, M.; Khademzadeh, A. Flow Control and Scheduling Mechanism to Improve Network Performance in Wireless NoC. *IET Commun.* **2020**, *14*, 2231–2239. <https://doi.org/10.1049/iet-com.2019.1033>.
50. Malikov, M.M.; Romanov, A.Y. Traffic Patterns in Networks-on-Chip: A Survey. *IEEE Access* **2025**, *13*, 148803–148815. <https://doi.org/10.1109/ACCESS.2025.3601971>.
51. Soni, N.; Deshmukh, K. Comparison Between Three Different Types of Routing Algorithms of Network on Chip. In *Advances in Optical Science and Engineering*; Lakshminarayanan, V., Bhattacharya, I., Eds.; Springer: New Delhi, India, 2015; pp. 447–459.
52. Dehyadgari, M.; Nickray, M.; Afzali-kusha, A.; Navabi, Z. Evaluation of Pseudo Adaptive XY Routing Using an Object Oriented Model for NOC. In *Proceedings of the 2005 International Conference on Microelectronics, Islamabad, Pakistan, 13–15 December 2005*; IEEE: Piscataway, NJ, USA, 2005; pp. 204–208.
53. Glass, C.J.; Ni, L.M. The Turn Model for Adaptive Routing. *ACM SIGARCH Comput. Archit. News* **1992**, *20*, 278–287. <https://doi.org/10.1145/146628.140384>.
54. Chiu, G.-M. The Odd-Even Turn Model for Adaptive Routing. *IEEE Trans. Parallel Distrib. Syst.* **2000**, *11*, 729–738. <https://doi.org/10.1109/71.877831>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.