LIA: Latency-Improved Adaptive routing for Dragonfly networks

MARIANO BENITO, Computer Sciences, Barcelona Supercomputing Center, Barcelona, Spain and Computer Engineering and Electronics Department, Universidad de Cantabria, Santander, Spain
ENRIQUE VALLEJO, Computer Engineering and Electronics Department, Universidad de Cantabria, Santander, Spain
RAMÓN BEIVIDE, Computer Engineering and Electronics Department, Universidad de Cantabria, Santander, Spain

Low-diameter network topologies require non-minimal routing, such as Valiant routing, to avoid network congestion under challenging traffic patterns like the so-called adversarial. However, this mechanism tends to increase the average path length, base latency, and network load. The use of shorter non-minimal paths has the potential to enhance performance, but it may also introduce congestion depending on the traffic patterns. This article introduces *LIA* (Latency-Improved Adaptive), a routing mechanism for Dragonfly networks which dynamically exploits minimal and non-minimal paths. *LIA* harnesses the traffic counters already present in contemporary switches to determine when it is safe to shorten non-minimal paths and to adjust routing decisions based on their information about the network conditions. Evaluations reveal that *LIA* achieves nearly optimal latency, outperforming state-of-the-art adaptive routing mechanisms by reducing latency by up to 30% while maintaining stable throughput and fairness.

CCS Concepts: • **Networks** → *Network simulations*; **Data center networks**; *Topology analysis and generation*; **Routing protocols**; **Intermediate nodes;**

Additional Key Words and Phrases: Valiant routing, non-minimal adaptive routing, non-minimal path shortening, LIA, dragonfly

ACM Reference Format:
Mariano Benito, Enrique Vallejo, and Ramón Beivide. 2025. LIA: Latency-Improved Adaptive routing for Dragonfly networks. *ACM Trans. Arch. Code Optim.* 22, 1, Article 39 (March 2025), 26 pages. https://doi.org/10.1145/3711914

The research was mainly performed while R. Beivide was a visiting researcher at the Barcelona Supercomputing Center.
New Paper, Not an Extension of a Conference Paper.
This work has been supported by the Spanish Ministry of Science and Innovation under contracts PID2022-136454NB-C21 (MCIN/AEI/10.13039/501100011033/FEDER) and TED2021-131176B-I00, and the Barcelona Supercomputing Center under projects MEEP with grant agreement 946002 and DITREA with grant agreement CONSER02023011NG.
Authors' Contact Information: Mariano Benito, Computer Sciences, Barcelona Supercomputing Center, Barcelona, Spain and Computer Engineering and Electronics Department, Universidad de Cantabria, Santander, Spain; e-mail: mariano.benito@unican.es; Enrique Vallejo, Computer Engineering and Electronics Department, Universidad de Cantabria, Santander, Spain; e-mail: enrique.vallejo@unican.es; Ramón Beivide, Computer Engineering and Electronics Department, Universidad de Cantabria, Santander, Spain; e-mail: ramon.beivide@unican.es.

ACM 1544-3973/2025/03-ART39
https://doi.org/10.1145/3711914

1 Introduction

Low-diameter high-radix interconnection networks, such as Dragonflies [20] are used in some of the largest computers [6]. They offer a compelling combination of high scalability and low network latency. However, they are susceptible to congestion, particularly when exposed to challenging traffic patterns, commonly referred to as *adversarial*. These scenarios can lead to a significant concentration of traffic on a few specific network links. To address this issue, *non-minimal source-adaptive* routing mechanisms can be used, dynamically choosing between minimal and non-minimal paths for each packet. These mechanisms need to define two key aspects: first, how to *select* between minimal and non-minimal path, and second, the *specific non-minimal path* to be utilized. Typically, the *selection* relies on the *UGAL* mechanism [27], which compares congestion levels (estimated via next-hop buffer occupancy) in both types of paths. Additionally, the definition of *non-minimal paths* often relies on Valiant routing [29], which diverts traffic to an intermediate switch chosen randomly and then forwards it to its actual destination, using minimal paths for both stages.

The main drawback of Valiant routing is that it increases the average path length, base latency and overall network load. Several proposals [7, 26] have aimed to *shorten* non-minimal paths as long as it does not introduce network bottlenecks. However, these approaches operate independently of traffic-pattern estimations. Nonetheless, the utilization of shortened paths can lead to pathological congestion and unfairness at certain loads. As a result, most prior proposals have leaned towards using long non-minimal paths [16].

Proposals like TPR [13] incorporate traffic-pattern adversity estimations based on switch counters [1, 2]. These estimations are used to adjust the *UGAL* parameters, biasing path selection according to traffic behavior. This approach is very promising, given the widespread availability of switch counters in current products. However, as discussed later in this article, there are fundamental limitations in how these routing mechanisms leverage traffic counters for the two key aspects defined previously. First, optimal routing decisions should be based on *offered traffic* values, which represent traffic under the assumption of infinite network resources. However, proposed selection mechanisms rely on *accepted traffic* estimations, which depends on previous routing decisions. These two concepts differ when the network becomes saturated, leading to suboptimal results in terms of maximum network throughput. Second, traffic counters hold valuable information about potential congestion points in the network. This information can be used to shorten non-minimal paths without introducing additional congestion points. Neglecting such data results in unnecessary network latency and longer paths.

Based on this analysis, this article introduces *LIA*: Latency-Improved Adaptive routing. This routing mechanism for Dragonfly networks relies on *UGAL*, which selects between minimal or non-minimal routing paths, and a limited set of local traffic counters at each switch, to determine when it is safe to shorten non-minimal paths.

To achieve this, *extended* traffic counters are used to track offered traffic, as opposed to accepted one, ensuring stable throughput even under saturation loads. Additionally, leveraging a *regular* global link arrangement, two small sets of *extended* counters are computed per switch: *global* and *remote*. *Global* counters serve a dual purpose: they help modulate the selection between minimal and non-minimal paths and determine when it is safe to skip the first local hop of non-minimal paths. Similarly, one or two *remote* counters are used to ascertain when it is safe to bypass the second local non-minimal hop, which occurs within a remote group. An overview of the system is presented in Figure 1.

Overall, *LIA* represents the first adaptive routing proposal for Dragonfly networks that combines the advantageous features of source routing, local information utilization, stable saturation

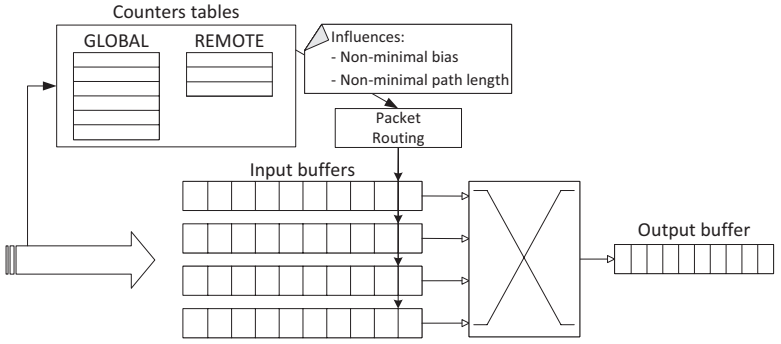


Fig. 1. LIA overview showing counters tables and its impact on packet routing decisions.

throughput, and the utilization of non-minimal paths with only the necessary hops to prevent congestion. Evaluation results demonstrate that LIA offers the lowest latency across the all load range and achieves maximum throughput without encountering congestion issues post-saturation, comparable to the best reference routing in each scenario.

The main contributions of this article are the following:

— We introduce extended counters, which track offered traffic, and global and remote counters, which estimate the impact (i.e., occurrence of congestion) of using short non-minimal paths.
— We present *LIA*, a source-based adaptive routing mechanism based on UGAL and the former counters, which identifies the traffic adversity and selects the most suitable path length, minimizing latency.
— We show a thorough evaluation of *LIA*, compared to other oblivious and source-adaptive routing mechanisms. Results show that *LIA* improves latency results under all evaluated traffic patterns, presents stable throughput, avoids unfairness and reduces latency by up to 30% compared to other source-adaptive routings.

2 Background

2.1 Dragonfly Topology

The Dragonfly [20] is a direct low-diameter topology deployed hierarchically in two levels, enabling it to adapt effectively to the system's packaging. Figure 2 shows a block diagram of a network with 5,256 compute hosts using this topology. The first level comprises groups of switches interconnected following certain *intra-group* topology by *local* communication links (L). These groups are connected by *global* links (G), according to an *inter-group* topology.

Topology parameters include the number of compute hosts connected to each switch p, the number of switches per group a, and the number of global links per switch h. Hence, the number of ports per switch, or switch radix k, must be $k \geq h + p + a - 1$. Each group has $a \cdot p$ injectors and $a \cdot h$ global links.

Arbitrary networks can be used in both topological levels; we consider *canonical* Dragonflies with a fully-connected graph in both levels and diameter three: one hop in each group plus one hop in the global interconnect. The relation $a = 2p = 2h$ [20] ensures a balanced use of the links under a load-balanced traffic. This equals the number of global links departing from a group to the number of terminals (or injectors). With a single link between pairs of groups, the network comprises up to $g = 1 + a \cdot h = 2h^2 + 1$ groups. For example, the $h = 6$ Dragonfly in Figure 2 has 6 terminals per switch, 12 switches per group and up to 73 groups, for an overall 5,256 compute hosts.

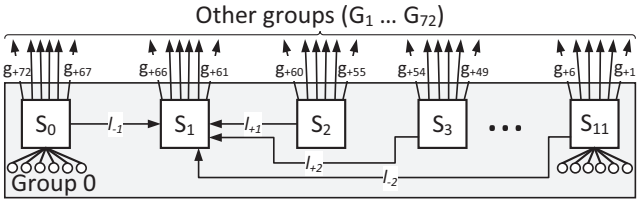


Fig. 2. Diagram of a Dragonfly with $h = 6$ and $t = 1$ and the notation used in this work.

Multiple global links (*global trunking*) increase global bandwidth between groups, but reduce the maximum number of groups to $g = 1 + a \cdot h/t = 2h^2/t + 1$ with t links between groups. For simplicity, we consider that the global links per group, $a \cdot h$, is multiple of t.

Considering the previous balance condition, we base our notation on the parameter h. We denote the switches in the group $S_0, S_1, \ldots, S_{2h-1}$. We denote with l_j ($\{l_{\pm 1}, l_{\pm 2}, \ldots, l_{\pm h/t}\}$) the local link *entering* switch S_i from S_{i+j} (modulo $2h/t$). Note that $l_{+h/t}$ and $l_{-h/t}$ both denote the same offset, and that local link l_{+i} is denoted l_{-i} in the opposite direction. There are $2ht$ links of each type l_i per group, t entering and leaving from each switch. Figure 2 depicts part of the $2h - 1$ local links entering switch S_1.

We consider a Palmtree global link arrangement [10]: without trunking, switch S_j ($0 \leq j < 2h$) in group i connects to the h switches S_{2h-1-j} in groups $i + h \cdot j + 1, i + h \cdot j + 2, \ldots, i + h \cdot j + h$ (modulo $2h^2 + 1$), this is, h consecutive global links per switch. This is partially depicted in Figure 2. With trunking, our arrangement also connects consecutive groups in each switch: switch $S_{j+k \cdot t}$ in group i connects to the h switches $S_{2h-1-j-k \cdot t}$ in groups $i + h \cdot j + 1, i + h \cdot j + 2, \ldots, i + h \cdot j + h$ (modulo $2h^2/t + 1$), with $j < a/k; 0 \leq k < t$.

2.2 Dragonfly Routing

This section presents first the basic minimal, non-minimal and adaptive routing mechanisms. Next, it introduces two improvements that are key to our proposal: shortened Valiant paths and counter-based adaptive routing.

2.2.1 Basic Routing. ***Minimal (MIN)*** routing consists of up to three hops: a *local* hop in the source group, a *global* hop to the destination group, and a *local* hop to the destination switch. We denote such path *LGL*, or L_1GL_2 to refer to each hop. Shorter paths may occur, depending on the location of the source and destination switches, such as *LG*, *GL*, *G*, and *L*.

Under adversarial traffic patterns (discussed in Section 2.3), *MIN* saturates some network links, which become a bottleneck. ***Valiant Load Balancing (VLB)*** [29] avoids such bottlenecks by randomizing traffic. Valiant routing first sends traffic minimally to a random intermediate switch (denoted S_{ROOT}) in Phase A; this path is denoted Path A and follows the sequence *LGL*. In Phase B, traffic is sent minimally from S_{ROOT} to the destination, following another sequence *LGL*. The complete path may comprise up to six hops, *LGL-LGL* (or $L_1G_1L_2 - L_3G_2L_4$). We consider four versions of *VLB* with different Path A lengths: *LGL, LG, GL, and G*. An example of three of them is presented in Figure 3.

Source-adaptive routing selects between minimal and Valiant routing at source router. Multiple alternatives have been proposed. *UGAL* [27] selects the path at injection based on the next hop buffer occupancy in each path (Q_{MIN} and Q_{VLB}, for each buffer occupancy, respectively, in phits). In the Dragonfly [17], *UGAL* routes a packet minimally when

$$Q_{MIN} \leq 2 \times Q_{VLB} + T. \tag{1}$$

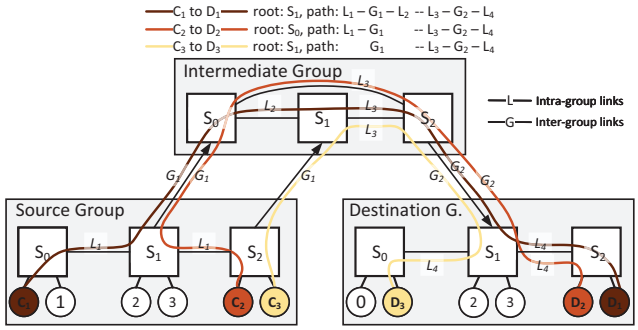


Fig. 3. Three different length *VLB* routing examples identifying the source, intermediate, and destination groups.

The threshold T can bias *UGAL* towards minimal or Valiant routing and although it is typically a constant, it can also be calculated dynamically [13, 19]. This selection can be improved using remote information [9, 17, 30], but this makes the routing more complex. Alternative mechanisms also support in-transit non-minimal adaptive routing [16–18].

2.2.2 Valiant Path Length. While Valiant routing prevents network bottlenecks, it employs paths that may be unnecessarily long and imposes an excessive load on the network. Shorter paths may be generated using *Restricted Valiant* [7], which employs a shortened version of Path A such as *LG*, *GL*, or *G*. Short non-minimal paths have been employed previously, such as the *LG* Path A in [17, 20]. However, under certain adversarial traffic patterns (studied in Section 3.2), it does not randomize traffic enough and certain links (local links in the intermediate group) become a bottleneck. Thus, the key goal is to find short Valiant paths that do not introduce pathological performance limitations.

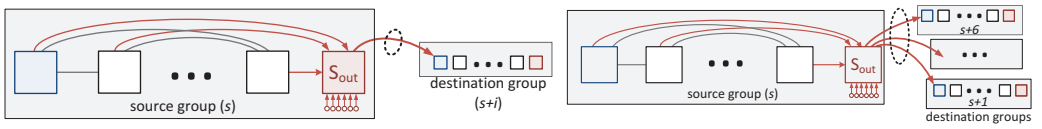
ACOR [7] dynamically adapts the length of non-minimal paths. Switches maintain an *ACOR level* that indicates the specific Path A used, following the sequence $G \leftrightarrow GL \leftrightarrow LGL$. This level increases or decreases based on an indirect estimation of congestion. While ACOR adapts Path A length, it requires that congestion propagates to the source and is agnostic to the actual traffic pattern, reacting only to the estimated traffic load.

2.2.3 Counter-based Adaptive Routing. ***Traffic pattern-based adaptive routing*** (***TPR***, [13]) improves *UGAL* by detecting the adverseness of the traffic pattern. *TPR* implements counters, similar to those available in current products [1, 2, 12], that measure both local and non-local intra-group and inter-group traffic forwarded through each port.

TPR defines multiple levels of traffic intensity, from "benign" to "adversarial". Each level corresponds to certain switch counters values, this is, the amount of traffic sent to a certain destination group. Thresholds that separate consecutive levels are obtained empirically. When sending traffic to a given destination, *TPR* modulates *UGAL* parameters based on the intensity level associated to such destination, so that under intense adversarial traffic patterns it is more likely to forward traffic non-minimally and vice-versa.

2.3 Traffic Patterns

This section presents different traffic patterns that have been identified to cause distinct levels of adverseness. Under ***random uniform (UN)*** traffic, the destination of each packet is uniformly selected among all nodes in the network. *MIN* routing is appropriate for such traffic, since it naturally balances traffic over all the network links.



(a) *ADV+1.* (b) *ADVc.*

Fig. 4. ADV+i and ADVc in a Dragonfly network with $h = 6, t = 1$. On the left, traffic from each source group s goes to group $s + i$ through t global links and on the right, it goes to the next $h = 6$ consecutive groups through the circled links.

Figure 4(a) presents ***adversarial shift (ADV+i)*** traffic, in which all nodes in each group send their traffic to a destination group placed i groups away. The t global links between these two groups become a bottleneck under *MIN* routing, limiting throughput to $t/a \cdot h = t/2h^2$ (note $t = 1$ in Figure 4(a)). To avoid it, non-minimal routing should be employed. Valiant routing, explained in Section 2.2.1, randomizes traffic and avoids jams. However, the additional hops of Path A, where packets are diverted to a random S_{ROOT}, increase the average path length (up to $LGL - LGL$) and base latency. The adverseness of *ADV+i* with different versions of Path A is discussed in Section 3.2.

Figure 4(b) depicts an example of ***adversarial-consecutive*** (***ADVc***) traffic [14], in which all nodes of a source group (s) send traffic to the h consecutive groups ($s + 1, s + 2, \ldots, s + h$, mod $2h^2 + 1$). Minimal paths concentrate traffic in t ($t = 1$ in the figure) switches denoted S_{out} in the source group, complicating congestion detection and limiting throughput to $t/(2h)$. While global links are not as much saturated as in *ADV+i*, *ADVc* introduces fairness issues, since nodes in S_{out} and nodes in different switches observe very different local traffic conditions.

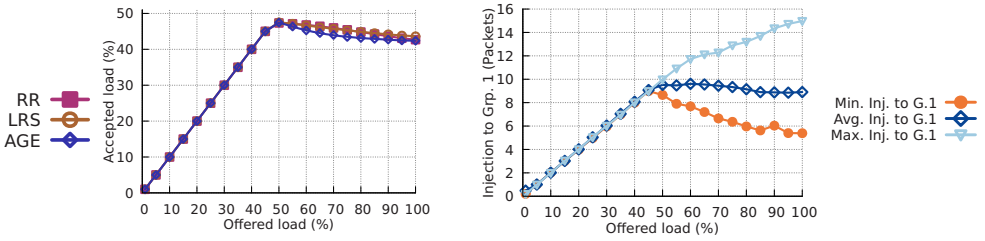
3 Analysis and Motivation

This section studies limitations of prior works: counter-based traffic detection and impact of short non-minimal paths.

3.1 Traffic Counters Measure Carried Traffic

TPR [13] relies on multiple traffic counters to modulate the selection between minimal and non-minimal routing at the source. Some counters track the amount of packets for a given destination group that are forwarded from the injection ports, per interval. These counters do not measure *offered traffic* (the one that would be sent in a network with infinite resources), but *carried traffic*, which is influenced by the routing mechanism. This cyclic dependency may lead to misleading results in the adaptive routing mechanism at saturation, explained in the following example.

Figure 5 shows the throughput vs. applied load plot for a Dragonfly under *ADVc* traffic using *TPR* routing mechanism and different arbitration policies: **round-robin** (***RR***) as our baseline, **least-recently served** (***LRS***) and age-based (*AGE,* [3]). At saturation the switch queues may get completely full, transiently stalling forwarding. Then, traffic counters may fall below the "high intensity" threshold, mislabeling intensity as "medium" and biasing routing towards MIN routing. This increases the bottleneck problem and further reduces both injected traffic and the counters for the given destination group. Overall, this results in reduced throughput after the saturation point, as observed in Figure 5(a). This effect depends on the counter implementation, so it is observed in the three arbitration policies considered. This congestion is not uniform across all switches, as observed in Figure 5(b), which depicts maximum, average and minimum traffic sent to group 1 per switch using the *RR* arbitration policy that allows us to see the variability. The difference between the highest and lowest injecting nodes grows with the offered load after saturation, leading to a significant imbalance.



(a) Average throughput obtained using different arbitration policies.

(b) Minimum, average and maximum traffic injected towards Group 1 using the baseline RR arbitration.

Fig. 5. Average results for switches in Group 0 under *ADVc* traffic using *TPR* in a Dragonfly with $h = 6$.

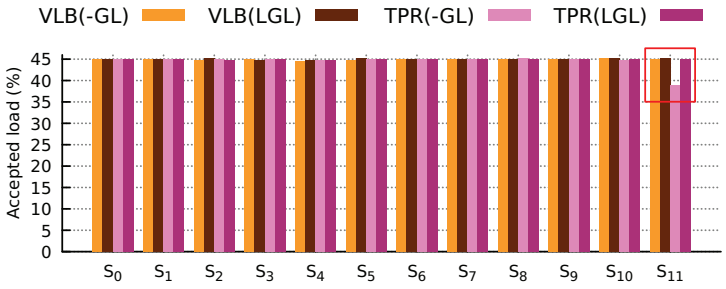


Fig. 6. Accepted throughput for each switch of group 0 under *ADVc* traffic with offered load 45%, comparing two path lengths using oblivious *VLB* and adaptive *TPR*.

3.2 Impact of Valiant Phase A Path Length

This section explores the impact of the length of Path A in a Dragonfly. If both source and destination were in the same group, Path A would be a single local hop L, but we focus on global traffic. When the destination is in a remote group, the non-minimal global hop G is clearly required to remove the bottleneck in the global link otherwise caused by adversarial traffic. Therefore, four paths for Valiant Phase A may be considered: *-GL*, *LGL*, *LG-*, and *-G-*. Note that with two global hops in the path, throughput is limited to 50%. Next subsections analyze the impact of saving each local hop in Path A by selecting an appropriate S_{ROOT}.

3.2.1 Impact of the First Local Hop in Valiant Phase A. Unbalance may occur in a group when a switch has to forward traffic received from its neighbors. We analyze this case using *ADVc* traffic. Figure 6 shows throughput per switch in a group under *ADVc* traffic, using oblivious VLB and adaptive *TPR*, with Valiant paths *-GL* and *LGL*. Using VLB, all injectors randomize traffic and all their traffic is accepted with either Path A, *-GL* or *LGL*. With adaptive *TPR*, part of the traffic from switches 0–10 is sent minimally to switch 11 (S_{out}) according to the *ADVc* pattern in Figure 4(b). This traffic is forwarded by the h global links in switch 11. With the *-GL* path, S_{11} employs these same h global links for traffic from its own h injectors, so injection in S_{11} is reduced causing unfairness. By contrast, when using *LGL* S_{11} may employ all the global links in the group, avoiding unfairness.

3.2.2 Impact of the Second Local Hop in Phase A. We consider here an homogeneous traffic, in which all switches inject with the same pattern. This traffic can be seen as the sum of multiple *ADV+i* traffic patterns, one per global offset in the original pattern. We analyze the congestion generated by each *ADV+i* traffic independently. Figure 7 shows accepted load under *ADV+i* traffic for

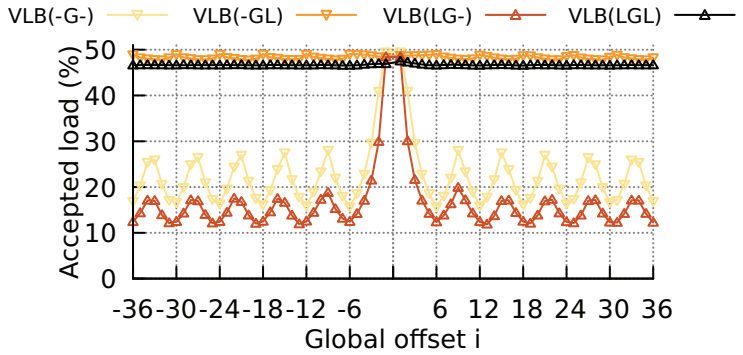


Fig. 7. Throughput with offered load of 50% on *ADV+i* in a Dragonfly using Valiant routing with different path lengths in Phase A.

each possible i in an $h = 6, t = 1$ canonical Dragonfly with an offered load of 0.5 phits/node/cycle and VLB. There are $g = 2h^2/t + 1 = 73$ groups, so $i \in \{\pm 1, \ldots, \pm 36\}$, in abscissas. Lines depict measured throughput for each Path A. Results for *TPR* adaptive routing, omitted for brevity, are similar.

Eliminating the hop L_2 significantly reduces throughput: both *GL* and *LGL* almost reach the maximum throughput 0.5, but shorter paths *G* and *LG* fall significantly lower. This motivates the use of *G* or *LG* when the load is low to reduce latency, and *GL* or *LGL* when the load is above a given threshold, which depends on the offset i, to avoid network saturation. The threshold varies with the offset of the *ADV+i* pattern, with minimum values for i being multiples of h, and maximum values for intermediate values of i.

We introduce an approximate model to determine this threshold without trunking ($t = 1$), and discuss $t > 1$ at the end. Consider $ADV + (k \cdot h + m)$ traffic with $0 \leq m < h$ and $k \geq 0$; negative values are symmetric. With the Palmtree arrangement employed in *LIA*, traffic received in the intermediate group through switch S_i needs to leave to the destination group through switch S_{i+k} or S_{i+k+1} $(mod\ 2h)$. Therefore, the traffic received in switch S_i is forwarded by only two local links l_k and l_{k+1} in the hop L_3. Specifically, flows received by m global links leave through local link l_{k+1} and the remaining traffic from $h - m$ links is forwarded by local link l_{+k}.[1] The largest of m and $h - m$ determines which type of local link saturates first and upper bounds accepted load, since both l_k and l_{k+1} can only accept 1 phit/node/cycle. $ADV + (k \cdot h)$ $(m = 0)$ provides the lowest throughput since local link l_k receives the aggregated throughput of all the h incoming global links in S_i, limiting throughput to $1/h$ on average. The highest throughput is obtained when the global offset is the intermediate value between $k \cdot h$ and $(k + 1) \cdot h$ because traffic is evenly distributed between both local links. Note also that $k = 0$ implies that part of the traffic employs l_{+1} and part of the traffic leaves directly without any local hop (there is no l_{+0}), reducing congestion.

Figure 8 depicts the example of $ADV + 8 = ADV + (1 \cdot h + 2)$ traffic using a G Valiant Path A in an $h = 6$ Dragonfly. Traffic from $m = 2$ global links is forwarded through local link l_{+2} whereas traffic from $h - m = 6 - 2 = 4$ global links is forwarded through local link l_{+1}. This limits throughput to $1/4 = 25\%$ under this traffic. Results in Figure 7 are close to this limit, with the difference owing to additional local hops in the source and destination groups.

[1]When the intermediate group is between the source and destination groups, this result is displaced by one unit. This tiny detail has been deliberately ignored in this analysis because it would make the model more complex with a negligible impact on performance. However, it explains the tiny drift of the minimal throughput values observed in Figure 7 for large offsets.

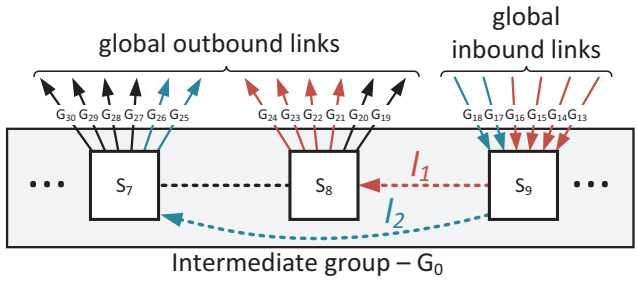


Fig. 8. Bottleneck at local links of intermediate group under *ADV+8* traffic pattern in a Dragonfly with $h = 6$ and $t = 1$.

With trunking $t > 1$ there are t times less intermediate groups, $2h^2/t$, so each group receives t times the traffic load from each source group. With the global link arrangement in Section 2.1, each intermediate group receives traffic through t different switches, so the load per switch remains. However, each switch has t local links of each type l_i, so the pathological congestion is spread through t links, reducing its impact by a factor of t. This implies that a network with trunking $t > 1$ employs a threshold t times higher, allowing for a wider load range in which the second local hop L_2 is omitted from Valiant paths without introducing pathological congestion.

4 Latency-Improved Adaptive (LIA) Routing for Dragonfly Networks

This section introduces the *LIA* routing for Dragonfly networks. After an initial overview, it details extended counters and non-minimal path selection.

4.1 LIA Overview

LIA implements source adaptive routing. First, source switches select appropriate intermediate nodes S_{ROOT} based on a restricted version of Valiant routing, which may skip each of the two local hops L_1 and L_2 in their Path A. Next, the selection of a minimal or non-minimal path relies on the occupancy of each path, based on a variant of *UGAL*. In both cases, decisions depend on an estimation of the current *offered* traffic, which relies on traffic counters.

Traffic counters combine information from injected traffic per interval and instantaneous amount of traffic in injection queues, as explained in Section 4.2. Since they extend the simple implementation of a traffic counter, these are denoted *extended* counters. *LIA* employs two sets of extended counters per switch: *global* and *remote* counters. *Global counters* are employed to determine whether it is safe to skip the first local hop L_1 and to modulate *UGAL*, whereas *remote counters* are used to skip L_2. Counter values are calculated locally in each switch; no counter information is distributed.

First, $2h^2/t$ *global counters* $\{G_{\pm 1}, G_{\pm 2}, \ldots, G_{\pm h^2/t}\}$ (one for each remote group) per switch measure the offered traffic towards each of the $2h^2/t$ remote groups. Their values are obtained from the switch injection ports. Source switches determine when to skip L_1 based on these counters, as defined in Section 4.3.1.

Second, $2h/t$ *remote counters* $\{R_{\pm 1}, R_{\pm 2}, \ldots, R_{\pm h/t}\}$ (one per type of local link, with the terminology introduced in Section 2.1) estimate the load that would be received on the local links in intermediate groups if the non-minimal routing omitted the L_2 local hop (*-G-* or *LG-*). The value of each counter R_i represents the individual contribution of the switch traffic to the congestion in remote local links of type l_i.

A **Remote Congestion Threshold** (**RCTh**) is determined empirically, such that the links l_i saturate when all the switches reach the RCTh value in their counter R_i. When the R_i value exceeds

this threshold, the L_2 link is employed in Path A for all non-minimal traffic associated to the counter R_i (based on the destination group, as defined in Section 4.3.2), diverting traffic away from links l_i; otherwise, the L_2 link is skipped, shortening Path A. This avoids the contribution of the current switch to the load of links l_i, only when such contribution could congest the links. Note that the traffic pattern or load level often varies per switch: some switches might employ a shortened Path A without L_2 (because their corresponding R_i counter is below the threshold) while others employ the long version (because their R_i counter is larger). Each of these counters R_i is derived from the values of a subset of the *global* counters, as defined in Section 4.3.2. Section 6.1.2 presents an example of empirical threshold selection.

Based on the values of the global and remote counters, *LIA* determines a suitable S_{ROOT} for each packet, selected such that the Path A adapts to the determined non-minimal path. Finally, the value of the global counter associated with the destination of the packet, which represents traffic intensity, is used to modulate *UGAL*. It biases the result towards minimal or non-minimal routing by modifying the threshold parameter T in Equation (1). Three load levels are defined, each with its own threshold T.

4.2 Traffic Estimation Using Extended Counters

Section 3.1 identifies the limitations of estimating traffic using the amount of packets that are injected (forwarded) from each input injection queue. Extended counters consider both newly *injected* packets (inj_i) and packets transiently *stored* in input injection queues ($stor_i$). The average number of stored packets is very small before the saturation point, and grows quickly when the load exceeds this point, because traffic cannot be delivered as fast as it is generated and packets get stalled in the input queues.

Extended counters combine information from both *injected* and *stored* packets. The dynamic range of the original *injection*-based counters depends on the sampling interval. By contrast, the range of *stored* packets depends on the size of the input buffers. The calculation of extended counters relies on a parameter w used to weight the two types of counters, as follows:

$$extended_i = inj_i + w \times stor_i. \tag{2}$$

The parameter w needs to be set so that (i) uniform traffic under saturation is not confused with adversarial traffic, and (ii) adversarial traffic after saturation is not mistaken with medium or low-intensity traffic, as seen in Section 3.1.

4.3 Non-minimal Paths in LIA

This section presents the mechanisms used by *LIA* to determine when and how to shorten non-minimal paths, based on estimations of traffic.

4.3.1 Global Counters and First Local Hop. Section 3.2.1 discussed the necessity of the first local hop L_1 in non-minimal paths. In particular, consider a shortened path without L_1 ($-GL_2$ or $-G-$). Since the first non-minimal hop is global, the selection of the intermediate group is limited to the groups directly connected to the source switch by one of its global links. When minimal paths also employ these global links, there may be not enough resources to accommodate traffic from other switches, such as the minimal traffic from other links observed in Section 3.2.1.

To avoid this problem, *LIA* uses the hop L_1 when several global links in the source switch would receive too much load if MIN routing were used. This is obtained from the extended global counters discussed in Section 4.2. Specifically, each switch tracks the global counters associated to the h groups directly connected to it. When *several* counters (specified by a parameter Saturated Global

Counters) exceeds a Global Port Saturation Threshold, then L_1 is used. Otherwise, the non-minimal path intermediate group is restricted to directly connected groups (i.e., L_1 not used).

4.3.2 Remote Counters and Second Local Hop. This section details remote counters, which estimate the extent of intermediate-group pathological congestion with the current traffic, and how to use them to determine when it is safe to skip the hop L_2. Section 3.2.2 explains how skipping the hop L_2 in non-minimal paths concentrates traffic in certain local links used for the hop L_3, causing congestion. Specifically, for traffic with global offset $k \times h + m$, with $0 \leq m < h$, hop L_3 exclusively employs local links of type l_k or $l_k + 1$, proportionally to $(h - m)$ and m, respectively.

LIA employs $2h/t$ remote counters per switch, one per each minimum value in the -*G*- throughput line in Figure 7, i.e., each possible global destination offset $ADV + (k \cdot h + m)$ with $m = 0$. Each of these counters estimates the load imposed in one type of local link l_i, should the hop L_2 be omitted from a non-minimal path.[2]

A packet with global offset $k \times h + m$ contributes to congestion in links l_k and l_{k+1}. *LIA* accounts for it in both corresponding counters R_k and R_{k+1}, proportionally to their *probability* of use $(h - m)/h$ and m/h, respectively. Therefore, flows with different but relatively close global offset contribute to congestion in the same type of local links: All $2h - 1$ flows with global offset $(k-1) \times h + 1$ to $(k+1) \times h - 1$ contribute to congestion in links of type l_k, and have to be accounted for in the remote counter R_k.[3]

Global counters already track the offered load to each destination group (i.e., global offset). Therefore, remote counters are directly calculated from their values G_i, with the following operation:

$$R_k = \sum_{j=-(h-1)}^{+(h-1)} \left(\frac{h - |j|}{h} \right) \cdot G_{k \cdot h + j \ (mod\ 2h^2+1)}. \tag{3}$$

For example, in an $h = 6$ Dragonfly the derived counter R_3 is obtained from global counters as follows: $R_3 = \frac{1}{6}G_{13} + \frac{2}{6}G_{14} + \frac{3}{6}G_{15} + \frac{4}{6}G_{16} + \frac{5}{6}G_{17} + \frac{6}{6}G_{18} + \frac{5}{6}G_{19} + \frac{4}{6}G_{20} + \frac{3}{6}G_{21} + \frac{2}{6}G_{22} + \frac{1}{6}G_{23}$.

Remote counters are used to evaluate at injection if omitting L_2 from a non-minimal path is safe, or if L_2 should be included to avoid congestion, as follows. For each packet sent to a destination group with global offset $k \times h + m$, the source switch checks the two remote counters R_k and (if $m > 0$) R_{k+1} associated with the destination global offset. If neither of them exceeds an empirical threshold (RCTh), then pathological congestion is not considered relevant and L_2 is skipped from Path A; otherwise, L_2 is included in Phase A to avoid congestion.

5 Evaluation Methodology

This section presents the methodology used to evaluate LIA. Section 5.1 describes the simulation infrastructure, Section 5.2 the implementation of the routing mechanisms used and Section 5.3 the traffic employed.

5.1 Simulation Infrastructure

LIA and other reference routings have been implemented in *FOGSim*, an open-source phit level cycle-accurate interconnection network simulator. Our *FOGSim* models employ ***combined input-output queued*** (***CIOQ***) switches [24] operating at 1 GHz with RR arbitration for both input and

[2]Both minimums associated with global offsets $\pm h^2$ correspond to saturation of the local link l_{+h}; these offsets are consecutive due to the modulo function, and both appear because of the aforementioned case of choosing the intermediate group between the source and destination groups.

[3]These flows also affect counters R_{k-1} and R_{k+1}.

output arbiters. The minimum number of **virtual channels** (**VCs**) required has been employed to avoid routing deadlocks, following an increasing order to visit them [20].

We simulate a Dragonfly network with Palmtree global link arrangement with $p = h = 6$ hosts and global links per switch, $a = 12$ switches per group and $t = 1$. The network can be built with 24-port switches and comprises more than 5K hosts and almost 1K switches arranged in 73 groups.

We model HPC switches with a 90 ns port-to-port latency. The latency of network links is set to 15 ns for electrical and 150 ns for optical ones. Buffers are properly dimensioned for link length and we set a 2× internal speedup.

Each point of the results represents the average of ten simulations, in which network statistics are collected for 60k cycles after 60k cycles of warm-up. Unless otherwise noted, all experiments employ the parameters in Table 1.

5.2 Routing Mechanisms

Oblivious mechanisms, *MIN* and *VLB* routings employ 2/1 and 4/2 VCs in local/global ports respectively. These routing mechanisms provide the best performance under uniform or worst-case traffic patterns, respectively. Four variants of *VLB* with different Path A lengths are considered: *LGL, LG-, -GL, and -G-*.

Universal Globally-Adaptive Load-balanced Global (***UGAL-G***, [20, 27]) is a per-packet source adaptive routing that chooses between *MIN* and *VLB* paths by comparing the queue length of the corresponding global output ports, with a 2× penalty factor for *VLB* paths. Additionally, a constant T can bias the routing towards *MIN* or *VLB* to adjust performance for benign or adversarial traffic patterns. So, a packet is routed minimally if $Q_{MIN} \leq 2 \times Q_{VLB} + T$ is satisfied. Instantaneous knowledge of queue lengths for all global ports in the source group (including the ones in other routers) allows *UGAL-G* to balance the load in global ports. However, its implementation is impractical.

Piggyback (PB) routing [17] implements per-packet source adaptive routing, relying on state information for each global channel in the group. PB denotes a global channel g as saturated when $Q_g > F \times \bar{Q} + Z$, where Q_g is the queue occupancy of g, $\bar{Q}$ is the average occupancy of all the global channels of the switch, F is a tuning parameter and Z is a threshold to filter out transient variations in minimal paths. Saturation information is distributed among switches of the group. *PB* employs *LGL* for Path A and 4/2 VCs to avoid deadlocks.

Piggyback-ACOR (***PB-ACOR***) extends *PB* based on *ACOR* [7], defined in Section 2.2.2. *PB-ACOR* increases the length of Phase A paths when packets are blocked several times at the head of the injection buffers. The Phase A paths follow the sequence $-G- \leftrightarrow -GL \leftrightarrow LGL$. Two thresholds are used to increase (IT_1, IT_2) and decrease (DT_1, DT_2) the non-minimal path length. A hysteresis cycle (*HI*) avoids oscillations. The length is extended when the blocked packet counter exceeds the corresponding increase threshold, or reduced if it is lower than the decrease threshold after the *HI* interval.

Traffic Pattern-based Adaptive Routing (***TPR***, [13]) is based on *UGAL* and has been briefly explained in Section 2.2.3. Our implementation of *TPR* employs two load thresholds (LI_l and LI_h) and three traffic regions to modulate the *UGAL* threshold T, focused on global inter-group traffic. *TPR* requires 4/2 VCs, the same as *PB*. The *TPR*'s window size for each counter is 200 cycles and the counters are implemented using a ring buffer. The parameters employed (presented in Table 1) have been selected after sweeping different configurations for the best performance.

LIA routing employs the extended traffic counters explained in Section 4.2, implemented as a ring buffer with a window size of 200 cycles. The remote counters are updated each cycle according to Equation (3). *LIA* modulates the length of the *VLB* path of Phase A as described in Section 4.3, and employs the *UGAL* thresholds defined in Table 1 (same values as those in *TPR*),

Table 1. Simulation Parameters

	Parameter	Value
Network configuration	Total end terminals	$N = 5,256$ hosts
	Topology	Canonical Dragonfly
	Global link arrangement	Palmtree
	Switch degree	23 ports
	Global trunking	$t = 1$
	Link speed	200 Gbps
	Packet size	250 bytes
	Switch frequency	1 GHz
	Switching mechanism	Virtual Cut-through
	Arbitration policy	RR
	Internal crossbar speedup	2×
	Forwarding latency	90 ns
	Local/Global link latency	15/150 ns (3/30 m)
	Injection queue size	126 KBytes
	Local/Global transit queue size	18/45 KBytes
	Local/Global/Injection VCs	2/1/1 (MIN), 4/2/1 (Other)
	UGAL threshold constant	$T = 0$ flits
PB	Global link state calculation	$F = 120\%, Z = 5$
	UGAL threshold constant	$T = 0$ flits
PB-ACOR	PB routing parameters	$F = 120\%, T = 0, Z = 5$
	Switch hysteresis interval	$HI = 500$ ns
	First/Second increase	$IT_1 = 15, IT_2 = 50$
	First/Second decrease	$DT_1 = 5, DT_2 = 15$
TPR	History window	$HW = 200$ cycles
	Inter low/high injection thresholds	$LI_l = 3, LI_h = 5$
	UGAL threshold T values (packets)	benign=15, mix=-3, adv.=-15
LIA	History window	$HW = 200$ cycles
	Inter low/high injection thresholds	$LI_l = 3, LI_h = 5$
	UGAL threshold T values (packets)	benign=15, mix=-3, adv.=-15
	Counters mode	Extended, $w = 0.1$ (Equation (2))
	Saturated Global Counters	$SGC = 3$
	Global Port Saturation Threshold	$GPSTh = 5$
	Remote Congestion Th.	$RCTh = 20$

with the implementation introduced in Section 4.1. The w and $RCTh$ parameters are derived from the analysis in Sections 6.1.1 and 6.1.2, respectively, whereas SGC and $GPSTh$ are selected by systematically sweeping different values.

5.3 Traffic Patterns

We employ *UN*, *ADV+i*, and *ADVc* synthetic traffic patterns, and two application models, *All-to-all* and *Graph500*. In synthetic traffic each node injects packets following a Bernoulli process modulated by an injection probability.

The best and the worst cases for this topology are examined with *UN* and *adversarial shift* with offset i (*ADV+i*). According to the analysis in Section 3.2.2, two particular offsets have been considered: 1 and h, resulting in *ADV+1* and *ADV+6*, respectively. They represent the least and the most congestion-prone cases, respectively, with respect to the use of the second local hop in Path A.

ADVc traffic is employed because it presents throughput unfairness if nonminimal paths are not selected properly, since the bottleneck switch (S_{out} in Figure 4(b)) gets all its global links occupied by the traffic routed minimally from all other switches in the group. *ADVc* requires non-minimal routing, but is less adversarial in terms of throughput than *ADV+i* using *MIN* routing (limited to $h/(a \cdot p)$ and $1/(a \cdot p)$ phits/node/cycle, respectively).

All-to-all traffic models a collective communication where each host sends a message to every other host in the network. Our implementation follows a linear shift exchange pattern [25], implemented in multiple MPI frameworks. This pattern performs communication on $N-1$ consecutive rounds; node i in round j communicates with node $(i+j)(modN)$. Because all hosts use the same offset in each round, when using *MIN* routing the All-to-All traffic becomes a series of consecutive adversarial traffic patterns, with varying offset per round. To address this effect, we employ two different placements of tasks to nodes: consecutive and random. The random placement reduces congestion and presents significantly better performance under *MIN* routing.

Graph500 traffic is implemented according to the model introduced in [15] for the Graph500 benchmark [23]. This is a BigData benchmark that is based on the execution of a **breadth-first search** (**BFS**) over a graph. Graph500 distributes a graph between the compute nodes. In each round, the BFS algorithm explores one additional level in the graph, performing point-to-point communications with the required destination compute nodes, according to the distribution of the graph vertices. Overall, the communication pattern is uniform within each phase (level exploration), and an all-reduce operation is used at the end of each phase for synchronization. Following the notation in [15], we simulate a graph of scale $s = 20$ and edgefactor $f_e = 16$ with a coalescing size of $c_s = 256$, the maximum number of levels to explore is restricted to 5 and the number of edges connected to the root vertex is $d_r = 3$.

Halo 3D traffic implements a 27-point stencil discretization application model presented in [22], derived from common HPC physics workloads. This pattern appears in parallel applications such as Lattice **Quantum ChromoDynamics** (**QCD**). Following the details in [22], our implementation employs a random placement policy to assign stencil sub-cubes to network endpoints.

6 Results

6.1 Extended Global and Remote Counters

6.1.1 Extended global Counters in LIA. We explore and compare six counter models: *Injected* is a traditional implementation in which the counter tracks forwarded traffic; *Stored* only considers the packets that transiently stay in the input queues, and is only presented as a reference; and our *Extended* mechanism, using four different weight values $w = \{1, 0.5, 0.2, 0.1\}$ as defined in Equation (2). To isolate the impact of the counter model, all mechanisms rely on longest paths for non-minimal routing (pure Valiant with *LGL*) and global counters are used to modulate *UGAL*. Figure 9 shows the traffic counter value and throughput for *ADV+1, ADVc*, and *UN* traffic. *ADV+h*, omitted, is similar to *ADV+1*.

The *Injected* counter model reflects the limitations analyzed in Section 3.1: Over the saturation point, input queues get full because traffic cannot be forwarded as fast as it is generated and the *injected* counter value decreases. This situation causes an incorrect estimation of the traffic adversity, considering that the network load is less adverse and incorrectly biasing *UGAL* towards *MIN*. Hence, the saturation throughput starts to fall.

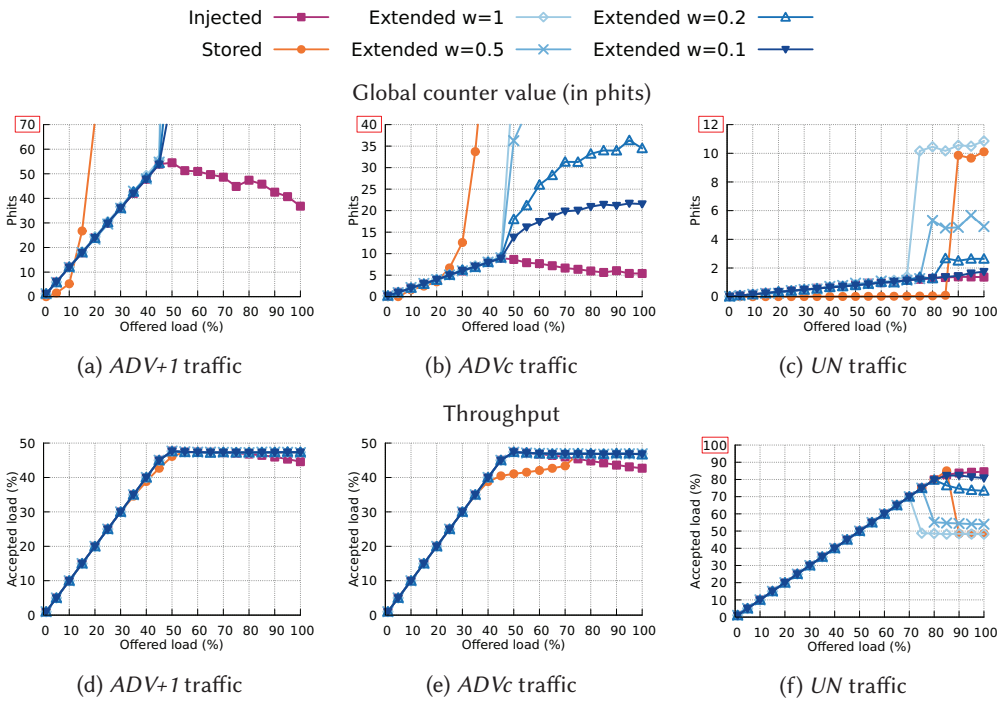
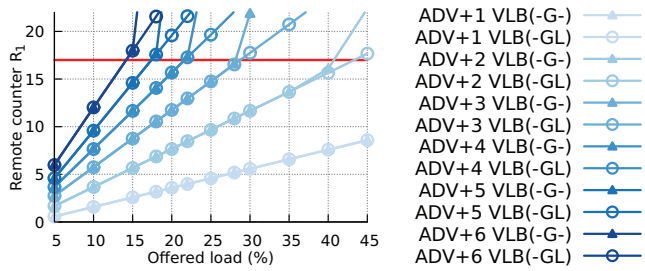


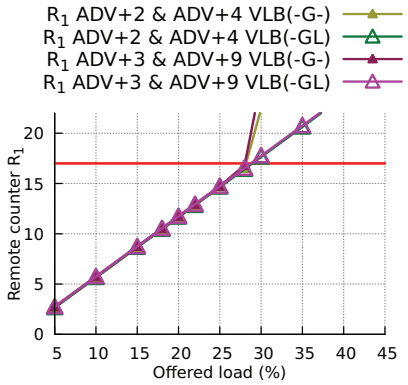
Fig. 9. Impact of extended counters. Global counter value (top) and throughput (bottom) for six implementations of traffic counters and three traffic patterns.

Regarding *stored* counters, prior to saturation nearly all packets are immediately forwarded resulting in an insignificant average value for the counter. This value grows quickly when the offered load exceeds the saturation point. This happens because packets cannot be delivered as fast as they are generated, so they are received in the buffer at the injection rate and are delivered at the accepted load rate, quickly accumulating at the injection queues. This has two problems for traffic estimation: first, it is difficult to differentiate *UN* traffic at saturation from adversarial traffic at medium or high loads, as observed in the poor throughput curves with severe congestion under *UN* after saturation; second, they only pass the traffic thresholds after the saturation point, but the routing should react before this point. Latency results (omitted for brevity) show poor results for *stored* counters under adversarial traffic.

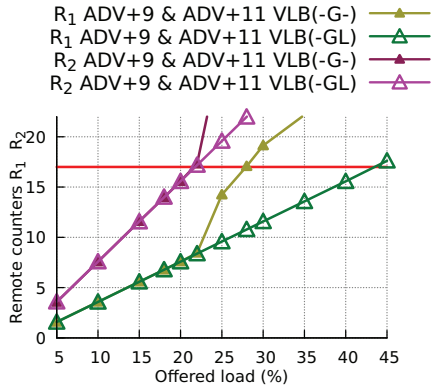
Extended counters combine *injection* and *stored* counters weighted by the parameter w, as explained in Section 3.1. Extended counters prevent the lack of information of using only the *injection* counter and avoid the problem caused on *UN* throughput by having only a *stored* counter. The weight w needs to be large enough to avoid congestion after saturation in adversarial traffic patterns, but small enough to prevent the congestion in *UN*. Figure 9 presents four options for tuning the parameter w. With large values $w = \{1, 0.5, 0.2\}$, the large value of the *stored* component produces the undesired behavior under *UN* traffic pattern, because counter values at saturation become too large to consider *UN* traffic as benign. Hence, the misled routing mechanism sends the traffic following *VLB* paths and as it can be seen in Figure 9(f), congestion appears. A small value $w = 0.1$ lowers the extended counter value to the same range of the *injection* counter under *UN* traffic, as seen in Figure 9(c). Hence, the throughput obtained in this case under *UN* is similar to using *injection* counters and the throughput under *ADV+1* and *ADVc*



(a) R_{+1} counter for ADV+i traffic pattern, $i \in \{1, ..., h\}$ with $\{-G-\}$ and $\{-GL\}$ Path A.



(b) R_{+1} counter value for two different pairs of combined adversarial traffic patterns.



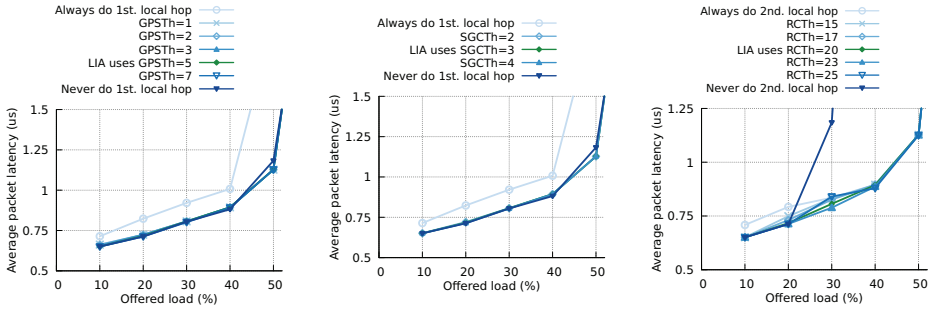
(c) R_{+1} and R_{+2} counters under combined adversarial traffic patterns ADV+9 and ADV+11.

Fig. 10. Remote counter analysis, using non-minimal paths *-G-* and *-GL*. Saturation using $\{-G-\}$ occurs for similar counter values in all cases, indicated by the red line. These sub-figures exemplify three different cases. Case (a) showcases isolated adversarial traffic patterns. Case (b) represents two traffic combinations in which the R_{+1} counter increases at the same rate in both mixes of traffic. The traffic mix in case c) affects two remote counters, which increase at different rates; the most restrictive one (R_{+2}, which is the first that crosses the red line), determines the need of the second local hop in the Path A of non-minimal routing.

remains stable after saturation thanks to the effect of the queued packets. Other values for the history window and buffer size may lead to different optimal values for w.

6.1.2 Remote Counters in LIA. This section explores how remote counters determine when the L_2 non-minimal hop is required as part of Path A. Figure 10 presents the evolution of the relevant remote counters for different traffic patterns and load. Two different non-minimal paths are considered: *-G-* and *-GL*, without and with the L_2 hop.

The first experiment in Figure 10(a) considers adversarial traffic pattern *ADV+i*, with different global offset i from 1 to $h = 6$. According to the analysis in Section 3.2.2 and the values in Figure 7, each of these traffic patterns gets a progressively lower saturation point when using Path A *-G-*, but *-GL* is always close to the 50% limit. This occurs because *-G-* concentrates traffic on one or two remote local links. Remote counters track the load in such links, so their value at saturation should be the same regardless of the traffic. Results confirm the previous analysis: all the different traffic patterns saturate at different loads, but the value of remote counters at the turning point is similar in all cases, around 17 phits/interval. Saturation is identified when counter values using



(a) *GPSTh* (b) *SGCTh* (c) *RCTh*

Fig. 11. Sensitivity analysis of different parameters of *LIA* under *ADVc* traffic.

-G- paths shoot up because input queues start to fill, but when using paths *-GL* they continue to grow steadily. Before this saturation point, counters for both paths match.

The application of Remote counters is not restricted to adversarial patterns with a single destination group. The two traffic patterns considered in Figure 10(b) divides traffic evenly between two destinations: groups +2 and +4 in one case and groups +3 and +9 in other case. In both situations the saturation occurs at the same turning point as before, around 17 phits/interval. Additionally, the R_{+1} counter value overlaps in both traffic mixes because when the non-null corresponding values are replaced in Equation (3) using $h = 6$, both cases match, since $R_{+1} = \frac{2}{6}G_2 + \frac{4}{6}G_4$ and $R_{+1} = \frac{3}{6}G_3 + \frac{3}{6}G_9$. Since traffic is divided evenly, $G_2 = G_4$ and $G_3 = G_9$, then R_{+1} grows at the same rate in both cases.

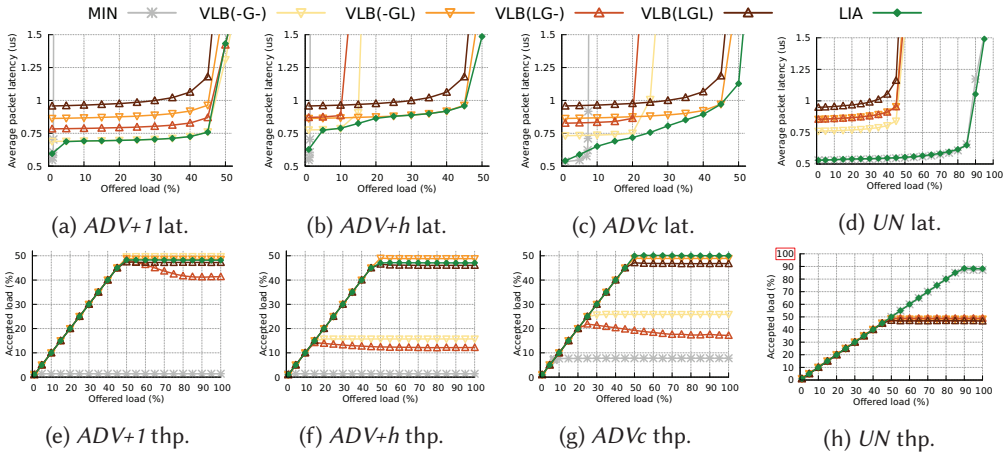
Finally, Figure 10(c) presents another situation where traffic is divided evenly between destination groups +9 and +11. Both values lie between $h = 6$ and $2h = 12$, so these flows increase remote counters R_{+1} and R_{+2} as follows: $R_{+1} = \frac{3}{6}G_9 + \frac{1}{6}G_{11}$ and $R_{+2} = \frac{3}{6}G_9 + \frac{5}{6}G_{11}$. Based on the previous equations, R_{+2} is expected to grow faster than R_{+1} because the traffic is distributed evenly to G_9 and G_{11}. As a result, the former is the counter that determines the saturation point of the path *-G-*. The results in Figure 10(c) confirm this, since R_{+2} grows faster and crosses the reference threshold at around 22% of the offered load. After this point, the values of both counters increase suddenly for paths *-G-*, since injection is restricted after local link l_{+2} saturates and packets accumulate at the input queues; this does not occur using paths *-GL*.

These results confirm that remote counters effectively point out the necessity of the L_2 local hop in Path A for non-minimal routing when any of the counter values crosses the defined threshold.

6.1.3 Sensitivity Analysis of Other Parameters in LIA. Figure 11 presents an analysis of the impact of three *LIA* parameters, using *ADVc* traffic. *GPSTh* and *SGCTh* are used to determine when to omit the first local hop in non-minimal paths. As observed in Figure 11(a) and (b), unless the parameters always force to use the first local hop, the impact of the parameter value is negligible. Figure 11(c) shows latency with different values of *RCTh*, which determine when to omit the second local hop. As observed in Figure 10, the value of this parameter should range around 17. The optimal latency in Figure 11(c) is obtained using $RCTh = 23$, but we select a more conservative value of $RCTh = 20$ to be on the safe side. Indeed, using a low value for *RCTh* is safe (it never introduces pathological congestion), but it may lead to using longer paths when it is not really necessary.

6.2 LIA Performance and Fairness

This section evaluates the performance of *LIA*, comparing to other oblivious and adaptive routings, and considering latency, throughput, and fairness.



(a) *ADV+1* lat. (b) *ADV+h* lat. (c) *ADVc* lat. (d) *UN* lat.

(e) *ADV+1* thp. (f) *ADV+h* thp. (g) *ADVc* thp. (h) *UN* thp.

Fig. 12. Average latency and throughput under *ADV+1*, *ADV+h ADVc*, and *UN* traffic patterns, comparing *LIA* with oblivious routings.

6.2.1 LIA Compared to Oblivious Routings. For the comparison, *MIN* has been selected as the baseline reference under *UN* traffic pattern because it achieves the best behavior on both latency and throughput. For *VLB* routing the four non-minimal paths considered in this work have been employed: -G-, -GL, LG-, and LGL. VLB(LGL) presents the best randomization to avoid any pathological congestion, but with the highest base latency. Figure 12 presents average latency and throughput of these mechanisms and the proposed *LIA* under different traffic patterns.

Performance results under *UN* traffic pattern are as expected: *VLB* with different paths present different base latencies and a throughput close to 50%. By contrast, *LIA* routing predominantly sends the traffic following minimal routes, like *MIN*, and achieves the same latency and throughput.

Using *MIN* under adversarial patterns, saturation is reached at very low load because global links become a bottleneck, and only a small part of the traffic can be delivered using minimal routes, specifically $1/(2h^2) \simeq 1.38\%$ and $1/(2h) \simeq 8.33\%$ under *ADV+i* and *ADVc*, respectively. Before these low saturation points, optimal latency is obtained using *MIN*. *VLB* improves these saturation values, to different levels depending on the non-minimal path length used. *LIA* raises that point close to 50% in all adversarial patterns, presenting the best average latency for the whole range of load.

Under *ADV+1*, *VLB* variants from 1 hop *-G-* to 3 hops *LGL* in Phase A obtain different base latencies, but the saturation point is near 50% in all cases. After this point, only VLB(LG-) suffers congestion. *LIA* obtains the best latency, like *VLB (-G-)* (Figure 12(a)) and a stable throughput very close to 50%. Under *ADV+h*, pathological congestion in local links limits throughput to $1/h \simeq 16\%$ for *-G-* and *LG-*. The two other *VLB* cases are near the 50% limit throughput. *LIA* throughput is close to the 50% limit and it achieves the best overall latency: at different increasing loads it approximates the latency of *MIN* up to 1.38%, VLB(-G-) up to 16.6% and VLB(-GL) up to 50%.

Results for *ADVc* follow the same trend as *ADV+h*. The saturation point is around 25% for *-G-* and *LG-*, and 50% with the second local hop. Again, *LIA* achieves the best latency because it adapts paths to the network conditions. Additionally, Figure 13 shows the average accepted load for each switch in group 0, under an offered load of 50% and *ADVc* traffic. VLB(-G-) and VLB(LG-), besides reduced throughput, suffer significant unfairness effects. By contrast, when *-GL* or *LGL* paths are used, *VLB* is fair, with a similar accepted load in all switches. *LIA* is totally fair, with the same accepted load in all switches.

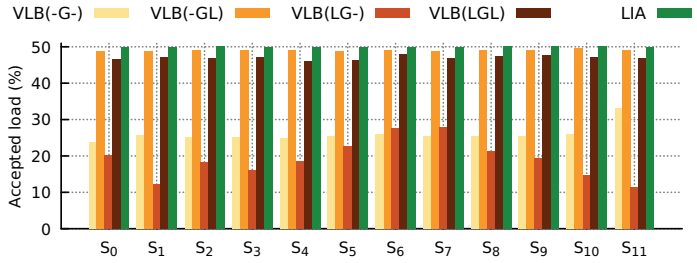


Fig. 13. Throughput accepted for each switch S_i of group 0 under *ADVc* traffic with an offered load of 50%, comparing *LIA* with oblivious routings.

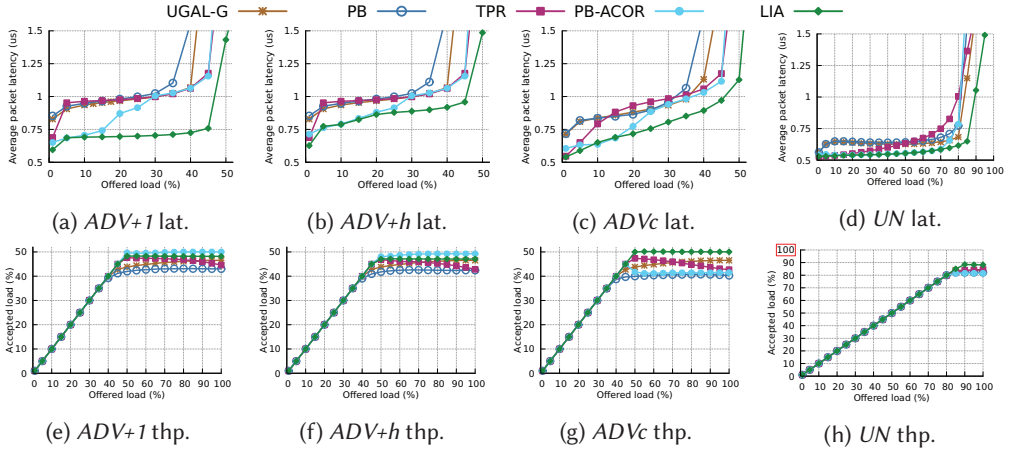


Fig. 14. Average latency and throughput under *ADV+1*, *ADV+h*, *ADVc*, and *UN* patterns, using *LIA* and other source-adaptive routings.

6.2.2 LIA Compared to Other Source Adaptive Routings. This section compares the performance of *LIA* with other source-adaptive routings: *PB* as the baseline reference; *UGAL-G* [20] as an ideal UGAL balancing the use of global ports; *TPR*, which relies on traffic counters and modifies the *UGAL* offset; and *PB-ACOR*, which blindly adapts non-minimal path to network load to reduce latency. Figure 14 presents the performance of these mechanisms under different traffics.

In terms of latency, *PB*, *UGAL-G* and *TPR* employ complete non-minimal paths, so under adversarial traffic patterns in Figure 14(a)–(c) they quickly catch up with the latency of *VLB(LGL)* presented in Figure 12. Even at a low load of 10%, LIA reduces latency by more than 25% over *PB*, *UGAL-G* and *TPR* under *ADV+1* traffic. *PB-ACOR* also improves latency at low loads, but it adapts the path length to absolute network load only, whereas *LIA* considers the detected traffic pattern and load. At an intermediate load of 30%, *LIA* reduces *PB-ACOR* latency by 30% under *ADV+1* traffic. Under *UN* traffic all routings are competitive.

In terms of throughput, *TPR* has a peak result close to 50% under adversarial traffic patterns in Figure 14(e)–(g). However, after this saturation point its throughput falls, as explained in Section 3.1, down to the result obtained by *PB*, which is the lowest in our evaluations. *PB-ACOR* and *LIA* obtain a stable throughput over saturation very close to the theoretical maximum of 50%. Again, under *UN* traffic all routings are competitive.

Figure 15 shows the average accepted load grouped by switch with an offered load of 50%. *PB* routing suffers from pathological unfairness under *ADVc*, as identified in [14]. *TPR* and *UGAL-G*

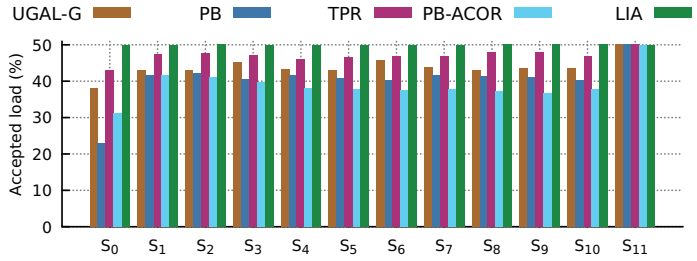


Fig. 15. Averaged throughput accepted for each switch of group 0 under *ADVc* traffic with an offered load of 50% comparing *LIA* with other source-adaptive routings.

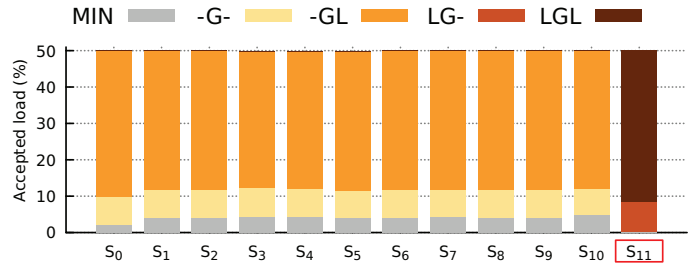


Fig. 16. Average accepted throughput using *LIA* for each switch in group 0, divided into MIN and the four possible misrouting policies. *ADVc* traffic with a load of 50%.

also exhibit this problem but it is less pronounced. Whereas *ACOR* adapting the non-minimal path length is fair [7], *PB-ACOR* inherits the unfairness issue of *PB* since it is based on the latter to select between minimal or non-minimal routes. As shown, *LIA* exhibits a perfect throughput fairness between all switches within the group.

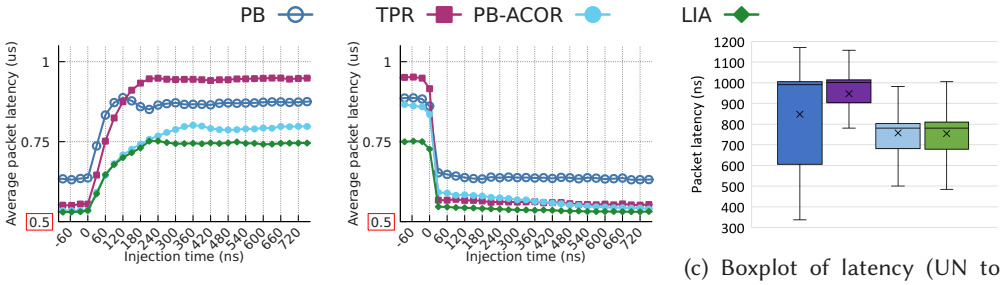
In conclusion, the best overall latency result is achieved by *LIA* and the throughput is very close to the theoretical maximum for each pattern.

6.2.3 Fairness and Use of the Hop L_1. This section analyses the behavior of *LIA* considering the use of the hop L_1, according to the implementation described in Section 5.2. Figure 16 breaks the accepted load for each switch by the path followed, either minimal or any of the non-minimal paths considered in this work.

Section 2.3 explains that *ADVc* unfairness comes from traffic concentrating in switch S_{11} (in our example) to be forwarded to the h consecutive groups. For this reason, short non-minimal paths that omit the hop L_1 should not be used by the terminals connected to the bottleneck switch S_{out}. This is solved by *LIA* as explained in Section 4.3.1. Figure 16 shows how S_{11} sends traffic using non-minimal paths with the hop L_1, because it identifies its global queues as saturated. Other switches continue using minimal or non-minimal paths without this hop L_1. In conclusion, *LIA* adapts the routing per switch to send the traffic minimally or non-minimally and to modify the length of the *VLB* Phase A path based on the network and particular switch conditions.

6.3 LIA Performance Under Transient Loads

Figure 17 shows the average packet latency under the transition between two different traffic patterns for the source-adaptive routing mechanism compared. It shows a change from *UN* to *ADVc* traffic patterns on the left and vice versa on the right. When the traffic pattern dynamically changes adversarially, from UN to ADVc in Figure 17(a), the latency of *UGAL-G* and *PB*



(a) UN to ADVc (b) ADVc to UN (c) Boxplot of latency (UN to ADVc).

Fig. 17. Average packet latency at $t = 0$ from *UN* to *ADVc* traffic patterns and vice versa in (a) and (b) and boxplot of packet latency at injection time equals to 210 ns of the transitory from UN to ADVc.

grows quickly because it increases the amount of non-minimally routed packets to adapt to the adversarial traffic pattern. The latency result of *TPR* before $t = 0$ explains that the amount of non-minimally traffic injected by TPR during the UN phase is lower than by PB. The same can be concluded for PB-ACOR and LIA. After that change, these routing algorithms start to route the packets through non-minimal paths, deduced from the increment on the average packet latency. LIA offers the best result, keeping the latency stable and as low as possible. Conversely, in Figure 17(b) the behavior is similar.

Figure 17(c) depicts the quartile boxplot for the latency of all the packets injected during the time interval from 181 to 210 ns, which is the range where the stabilization happens. The results show that *LIA* and *PB-ACOR* latency is quite lower than for the other routing mechanisms evaluated. Moreover, the amplitude between the 1st and 3rd quartile of *TPR*, PB-ACOR and LIA is significantly lower than in *PB*. However, the latency values reported by *TPR* are bigger, as seen in Figure 17(a).

6.4 LIA Performance with Multiple Traffic Patterns

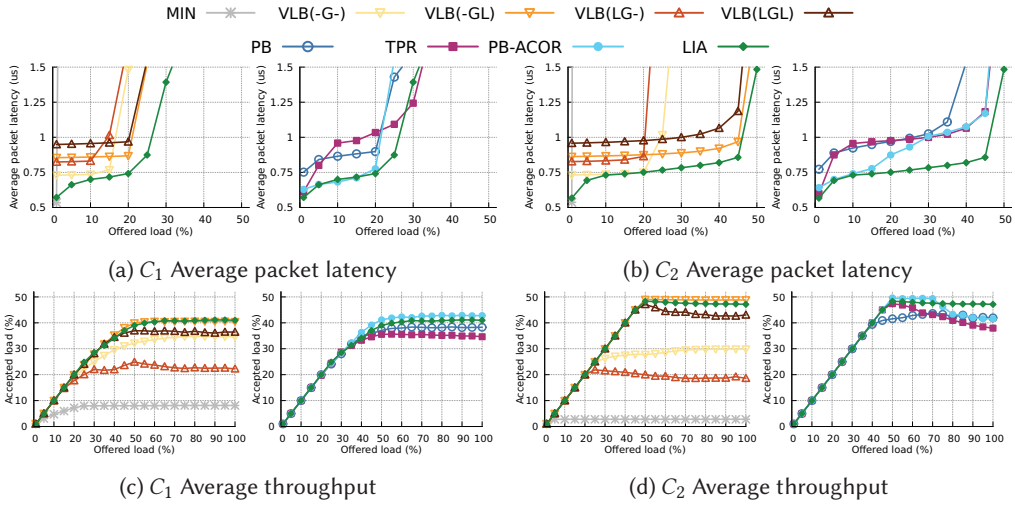
This section evaluates *LIA* using different traffic patterns in different areas of the network. Nodes in each network area only communicate with other nodes in the same area, following the traffic pattern of their area. Note that non-minimal routing can always divert traffic through other areas of the network. We have evaluated two traffic combinations: C_1 defines four areas, each with 25% of the nodes, and traffic patterns *UN, ADV+1, ADV+h*, and *ADVc*, respectively; C_2 defines two areas interleaving the nodes, so odd nodes send to *ADV+1* and even ones send to *ADV+h*.

Figure 18 shows average latency and throughput. These results are aligned with previous sections, in which *LIA* achieves the best or a very competitive performance. Additionally, they prove that *LIA*, using only information local to the switches, appropriately adapts path length even in presence of different traffic patterns and when a single group can't know about all traffic going to a group.

6.5 Application Execution Time

This section evaluates *LIA* using realistic application traffic patterns: *All-to-all*, with consecutive and random task placement, *Graph500* and Halo 3D. Tasks run to completion and performance is measured from the execution time. Values are normalized to the result of *MIN* routing in each case and Figure 19 presents the speedup of each routing mechanism.

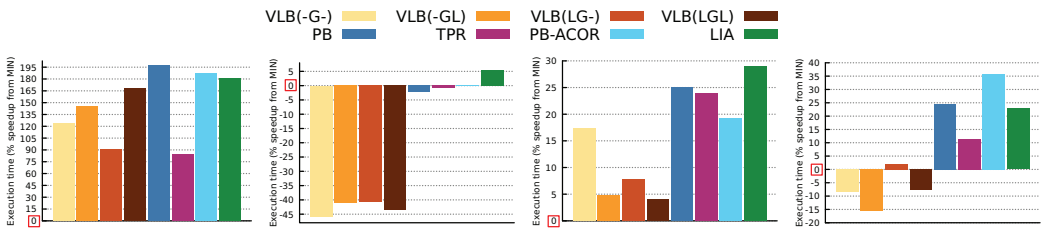
The consecutive task placement in Figure 19(a) is not well suited for *All-to-all*. With the linear shift exchange pattern, global links become bottlenecks and the *All-to-all* communication requires 350 972 cycles to finish with *MIN* routing. The four Valiant variants present a very significant



(a) C_1 Average packet latency (b) C_2 Average packet latency

(c) C_1 Average throughput (d) C_2 Average throughput

Fig. 18. Average latency (above) and average throughput (below) under C_1 : {*UN, ADV+1, ADV+h, ADVc*} and C_2 : {*ADV+1, ADV+h*} traffic cases comparing *LIA* with oblivious (left) and other source-adaptive (right) routings.



(a) *All-to-all-CTP* (b) *All-to-all-RTP* (c) *Graph500* (d) *Halo 3D*

Fig. 19. Execution time speed-up of oblivious and source-adaptive routing algorithms over *MIN* under *All-to-all*, using consecutive (*-CTP*) and random (*-RTP*) task placement, *Graph500* and Halo 3D traffic patterns.

speedup, by removing this bottleneck with the traffic randomization. Additionally, *PB* and *PB-ACOR* behave similarly and *LIA* achieves a competitive mark against them and considerably improves the result of *TPR* in both applications. By contrast, under random task placement in Figure 19(b), *MIN* performs nicely in *All-to-all*, reducing the completion time to 75 555 cycles. In this case, the randomization introduced by Valiant is counter-productive in both applications. In *All-to-all LIA* is the only routing mechanism that improves the result of *MIN*. Under *Graph500*, Valiant is effective and adaptive routings perform even better. *LIA* outperforms all other mechanisms, by at least 4% with respect to *TPR* and *PB*, which do not shorten non-minimal paths, or 10% with respect to *PB-ACOR*, which is not aware of the traffic pattern. Under *Halo 3D*, Valiant performs worse than MIN routing as expected based on its implementation although the evaluated adaptive routings outperforms oblivious algorithms by using non-minimal routes when those are required. *LIA* performs like *PB* and better than *TPR*.

7 Discussion

Is the number of counters required by* LIA *feasible? The number of counters required in *LIA* is affordable, corresponding to the number of groups plus local links per group. Considering the

Dragonfly-based Frontier supercomputer (currently in the Top-3), 73 global and 31 remote counters per switch would be required, which is affordable [1, 2]. Comparatively, current switches employ hundreds of traffic counters for routing and congestion control tasks [5, 12]. Note that these are traditional counters, not the extended counters that are introduced in this work, but its number clearly exceeds the requirements of *LIA*.

What are the implementation challenges of extended counters? Although the implementation of *LIA* counters requires to store more transient values for the history window ($HW = 200$ cycles in our evaluations, according to the parameters in Table 1), the amount of storage can be proportionally reduced if each transient value accumulates the traffic forwarded during multiple cycles, instead of saving one value per cycle. *LIA* leverages these counters for routing, but their calculation and distribution within the switch is not in the packet critical path: counters are updated periodically, and a small delay only affects the response to traffic changes, which is typically in the order of microseconds.

Does* LIA *need to adapt its parameters to different workloads? Tuning parameters depend on the network characteristics (network size, switch buffer sizes, history window, etc.) and not depend on the specific workload traffic. Hence, they do not need to be modified for different workloads.

What are the limitations of* LIA *with respect to a progressive adaptive routing? *LIA* is conservative and it only skips the second local hop (by selecting an appropriate intermediate Valiant node) when it is certain that doing it does not introduce congestion. However, this is estimated at the source switch based on its own counter values derived from its local nodes traffic, without remote information. Depending on the traffic pattern of other switches, there could be a situation in which a large remote counter suggests the use of long paths in a switch, but a shorter path would be safe (because other switches are not loading the corresponding local links). A progressive mechanism might re-evaluate this decision in the intermediate group, selecting the shortest possible path in all cases.

8 Related Work

Previous works have considered the use of traffic counters to modify routing based on a central controller, such as Hedera [4]. They work on a per-flow granularity and their adaptation time is significantly larger than in *LIA*. *LIA* employs extended traffic counters to estimate offered traffic instead of carried traffic. Hedera relies on an iterative algorithm instead. *TPR* [13] employs traffic counters to modify *UGAL* thresholds, but it fails to estimate offered traffic and does not shorten non-minimal paths.

Like *TPR*, our proposal *LIA* employs some counters and thresholds to adapt routing to the network traffic conditions. DGB [19] delegates the task of adaptive routing parameter tuning to a gradient descent optimization algorithm, and implements reinforcement learning techniques (specifically, the epsilon-greedy algorithm) to apply this mechanism. The application of reinforcement learning for parameter tuning in *LIA* could be considered as a future line of research.

Several mechanisms have proposed shortened variants of Valiant for the Dragonfly. The original proposal by Kim et al. in [20] selects a random intermediate *group*, instead of a *switch*, (LG-) with our terminology. Paths with a single non-minimal hop (-G-) have been used in previous work [8]. Both alternatives introduce pathological performance issues [16], analyzed in Section 3. *ACOR* [7] reduces the path length, but falls back to the complete path under high injection loads to avoid pathological issues. However, *ACOR* is oblivious to the traffic pattern and requires congestion to propagate to the source, leading to suboptimal results in certain cases. That work introduces

a *Restricted Valiant* mechanism employed here and the concept of restrict Valiant intermediate nodes is also employed in BoomGATE [21] to create subsets of *VLB* paths for deadlock avoidance. Similarly, *T-UGAL* [26] employs a subset of *VLB* paths with shorter average length. Then, it reduces the hops in both Phases A and B. However, it relies on an offline computation of paths, without adapting to the traffic pattern.

The use of short non-minimal Valiant paths has been also considered in different topologies. Yébenes et al. identified the *turn-around problem* [31] in Valiant in the Slim Fly, topology, and shortened the Valiant path to avoid it. However, they do not focus on other cases where path shortening improves performance and does not introduce congestion. Multiple short non-minimal paths are employed in other cases. For example, the Jellyfish [28] random topology relies on a *k-shortest path* algorithm [32] to find a set of paths (minimal and non-minimal) to each destination. However, it does not study path selection techniques, leaving that task to the congestion control mechanism. Moreover, it would never consider maximum-length Valiant paths, which are required in some cases as observed in this work.

Extended counters avoid congestion after saturation due to counter value miscalculation. However, a throughput drop can be observed in other cases such as oblivious Valiant. This effect, which has been observed before in the literature [11, 30], might have different root causes such as different load received on different network links. An analysis of the impact of the deadlock avoidance mechanism on these congestion issues is presented in [11].

9 Conclusions

Our adaptive routing *LIA* employs *UGAL* and introduces *extended* traffic counters, which leverage the Palmtree global link arrangement to estimate offered load and shorten non-minimal paths in Dragonfly networks.

The results show that *extended* counters correctly identify network traffic, leading to accurate routing under both uniform and adversarial traffic patterns. *LIA* performs equal to or better than the best oblivious routing for each traffic pattern analyzed. In those conditions, *LIA* also provides the lowest latency, given that non-minimal paths with less hops are used when VLB is required to avoid congestion. This feature allows latency reductions up to 30% over *PB* source-adaptive routing. Furthermore, its throughput is very close to the theoretical maximum for each traffic pattern analyzed, which makes it very competitive against state-of-the-art adaptive routings. Moreover, *LIA* does not present obvious effects of routing unfairness.

Acknowledgments

Authors thank the support of the European HiPEAC Network of Excellence.

References

[1] 2018. *About Aries Hardware Counters*. techreport. Cray Inc. Retrieved from https://support.hpe.com/hpesc/public/docDisplay?docId=a00113858en_us&page=About_Aries_Hardware_Counter_S-0045.html. Accessed: 2023-03.

[2] 2019. *InfiniBand Port Counters*. techreport. Mellanox. Retrieved March, 2023 from https://enterprise-support.nvidia.com/s/article/infiniband-port-counters

[3] Dennis Abts and Deborah Weisser. 2007. Age-based packet arbitration in large-radix k-ary n-cubes. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. 1–11.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. 2010. Hedera: Dynamic flow scheduling for data center networks. In *Conference on Networked Systems Design and Implementation*.

[5] R. L. Alverson, A. M. Bataineh, J. P. Beecroft, T. L. Court, A. M. Ford, E. L. Froese, D. C. Hewson, J. G. Kopnick, A. S. Kopser, D. Roweth, G. J. Faanes, M. F. Higgins, T. J. Johnson, T. A. Jones, W. J. Reinhard, S. L. Scott, and E. J. Turner. 2020. Switch Device for Facilitating Switching in Data-driven Intelligent Network. (2020). Patent No. WO/2020/236286, Filed Nov 26th., 2020.

[6] Scott Atchley, Christopher Zimmer, John Lange, David Bernholdt, Veronica Melesse Vergara, Thomas Beck, Michael Brim, Reuben Budiardja, Sunita Chandrasekaran, Markus Eisenbach, Thomas Evans, Matthew Ezell, Nicholas Frontiere, Antigoni Georgiadou, Joe Glenski, Philipp Grete, Steven Hamilton, John Holmen, Axel Huebl, Daniel Jacobson, Wayne Joubert, Kim Mcmahon, Elia Merzari, Stan Moore, Andrew Myers, Stephen Nichols, Sarp Oral, Thomas Papatheodore, Danny Perez, David M. Rogers, Evan Schneider, Jean-Luc Vay, and P. K. Yeung. 2023. Frontier: Exploring exascale. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC'23)*. DOI: https://doi.org/10.1145/3581784.3607089

[7] M. Benito, P. Fuentes, E. Vallejo, and R. Beivide. 2019. ACOR: Adaptive congestion-oblivious routing in dragonfly networks. *Journal of Parallel and Distributed Computing* 131 (2019), 173–188.

[8] Mariano Benito, Enrique Vallejo, and Ramon Beivide. 2015. On the use of commodity ethernet technology in exascale HPC systems. In *International Conference on High Performance Computing (HiPC'15)*.

[9] Mariano Benito, Enrique Vallejo, Cruz Izu, and Ramón Beivide. 2019. Non-minimal adaptive routing based on Explicit Congestion Notifications. *Concurrency and Computation: Practice and Experience* 31, 2 (2019), 1–27.

[10] C. Camarero, E. Vallejo, and R. Beivide. 2014. Topological characterization of hamming and dragonfly networks and its implications on routing. *ACM Transactions on Architecture and Code Optimization* 11, 4 (2014), 1–25.

[11] A. Cano, C. Camarero, C. Martínez, and R. Beivide. 2023. Analysing mechanisms for virtual channel management in low-diameter networks. In *International Symposium Computer Architecture and High Performance Computing (SBAC-PAD'23)*. 12–22.

[12] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler. 2020. An in-depth analysis of the slingshot interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage, and Analysis*.

[13] P. Faizian, J. F. Alfaro, M. S. Rahman, M. A. Mollah, X. Yuan, S. Pakin, and M. Lang. 2018. TPR: Traffic pattern-based adaptive routing for dragonfly networks. *IEEE Transactions on Multi-Scale Computing Systems* 4, 4 (2018), 931–943.

[14] Pablo Fuentes, Enrique Vallejo, Cristóbal Camarero, Ramón Beivide, and Mateo Valero. 2016. Network unfairness in dragonfly topologies. *The Journal of Supercomputing* 72, 12 (2016), 4468–4496.

[15] Pablo Fuentes, Mariano Benito, Enrique Vallejo, José Luis Bosque, Ramón Beivide, Andreea Anghel, Germán Rodríguez, Mitch Gusat, Cyriel Minkenberg, and Mateo Valero. 2017. A scalable synthetic traffic model of Graph500 for computer networks analysis. *Concurrency and Computation: Practice and Experience* 29, 24 (2017).

[16] M. García, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, G. Rodríguez, J. Labarta, and C. Minkenberg. 2012. On-the-fly adaptive routing in high-radix hierarchical networks. In *International Conference on Parallel Processing*.

[17] Nan Jiang, John Kim, and William J. Dally. 2009. Indirect adaptive routing on large scale interconnection networks. In *International Symposium on Computer Architecture (ISCA'09)*. 220–231.

[18] Y. Kang, X. Wang, and Z. Lan. 2021. Q-Adaptive: A multi-agent reinforcement learning based routing on dragonfly network. In *High-Performance Parallel and Distributed Computing (HPDC'21)*.

[19] Hans Kasan, Gwangsun Kim, Yung Yi, and John Kim. 2022. Dynamic global adaptive routing in high-radix networks. In *International Symposium on Computer Architecture (ISCA'22)*.

[20] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*. IEEE.

[21] G. Kwauk, S. Kang, H. Kasan, H. Son, and J. Kim. 2021. BoomGate: Deadlock avoidance in non-minimal routing for high-radix networks. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*.

[22] N. McDonald, M Isaev, D. Flores, A. Davis, and J. Kim. 2019. Practical and efficient incremental adaptive routing for HyperX networks. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC'19)*.

[23] R. Murphy, K. Wheeler, B. Barrett, and J. Ang. 2010. Introducing the graph 500. *Cray User's Group* (2010). https://www.osti.gov/biblio/1014641

[24] Balaji Prabhakar and Nick McKeown. 1999. On the speedup required for combined input- and output-queued switching. *Automatica* 35, 12 (1999), 1909–1920.

[25] Bogdan Prisacari, German Rodriguez, and Cyriel Minkenberg. 2013. Generalized hierarchical all-to-all exchange patterns. In *International Symposium on Parallel and Distributed Processing*. IEEE.

[26] M. S. Rahman, S. Bhowmik, Y. Ryasnianskiy, X. Yuan, and M. Lang. 2019. Topology-Custom UGAL routing on dragonfly. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC'19)*.

[27] A. Singh. 2005. *Load-balanced Routing in Interconnection Networks*. Ph.D. Dissertation. Stanford.

[28] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. 2012. Jellyfish: Networking data centers randomly. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*.

[29] L. G. Valiant and G. J. Brebner. 1981. Universal schemes for parallel communication. In *Annual ACM Symposium on Theory of Computing (STOC'81)*. ACM, 263–277.

[30] J. Won, G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott. 2015. Overcoming far-end congestion in large-scale networks. In *International Symposium on High Performance Computer Architecture (HPCA'15)*.
[31] P. Yébenes, J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, and T. Hoefler. 2017. Improving non-minimal and adaptive routing algorithms in slim fly networks. In *Symposium on High-Performance Interconnects (HOTI'17)*. 1–8.
[32] Jin Y. Yen. 1971. Finding the K shortest loopless paths in a network. *Management Science* 17, 11 (1971), 712–716.

Received 13 March 2024; revised 18 November 2024; accepted 10 December 2024