HopliteBuf: Network Calculus-Based Design of FPGA NoCs with Provably Stall-Free FIFOs

TUSHAR GARG, SAUD WASLY, RODOLFO PELLIZZONI, and NACHIKET KAPRE,
University of Waterloo, Canada

HopliteBuf is a deflection-free, low-cost, and high-speed FPGA overlay Network-on-chip (NoC) with stall-free buffers. It is an FPGA-friendly 2D unidirectional torus topology built on top of HopliteRT overlay NoC. The stall-free buffers in HopliteBuf are supported by static analysis tools based on network calculus that help determine worst-case FIFO occupancy bounds for a prescribed workload. We implement these FIFOs using cheap LUT SRAMs (Xilinx SRL32s and Intel MLABs) to reduce cost. HopliteBuf is a hybrid microarchitecture that combines the performance benefits of conventional buffered NoCs by using stall-free buffers with the cost advantages of deflection-routed NoCs by retaining the lightweight unidirectional torus topology structure. We present two design variants of the HopliteBuf NoC: (1) single corner-turn FIFO ($W \rightarrow S$) and (2) dual corner-turn FIFO ($W \rightarrow S + N$). The single corner-turn ($W \rightarrow S$) design is simpler and only introduces a buffering requirement for packets changing dimension from the X ring to the downhill Y ring (or West to South). The dual corner-turn variant requires two FIFOs for turning packets going downhill ($W \rightarrow S$) as well as uphill ($W \rightarrow N$). The dual corner-turn design overcomes the mathematical analysis challenges associated with single corner-turn designs for communication workloads with cyclic dependencies between flow traversal paths at the expense of a small increase in resource cost. Our static analysis delivers bounds that are not only better (in latency) than HopliteRT but also tighter by $2 - 3\times$. Across 100 randomly generated flowsets mapped to a 5×5 system size, HopliteBuf is able to route a larger fraction of these flowsets with <128-deep FIFOs, boost worst-case routing latency by $\approx 2\times$ for mutually feasible flowsets, and support a 10% higher injection rate than HopliteRT. At 20% injection rates, HopliteRT is only able to route 1–2% of the flowsets, while HopliteBuf can deliver 40–50% sustainability. When compared to the $W \rightarrow S_{bkp}$ backpressure-based router, we observe that our HopliteBuf solution offers 25–30% better feasibility at 30–40% lower LUT cost.

CCS Concepts: • **Hardware** → **Reconfigurable logic and FPGAs**; • **Networks** → **Network on chip**; • **Mathematics of computing** → **Calculus**;

Additional Key Words and Phrases: Network calculus, FPGA overlay NoC, stall-free buffers

ACM Reference format:
Tushar Garg, Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. 2020. HopliteBuf: Network Calculus-Based Design of FPGA NoCs with Provably Stall-Free FIFOs. *ACM Trans. Reconfigurable Technol. Syst.* 13, 2, Article 6 (February 2020), 35 pages.
https://doi.org/10.1145/3375899

Authors' addresses: T. Garg, S. Wasly, R. Pellizzoni, and N. Kapre, University of Waterloo, Ontario, Canada; emails: {t3garg, swasly, rpellizz, nachiket}@uwaterloo.ca.

1936-7406/2020/02-ART6 $15.00
https://doi.org/10.1145/3375899

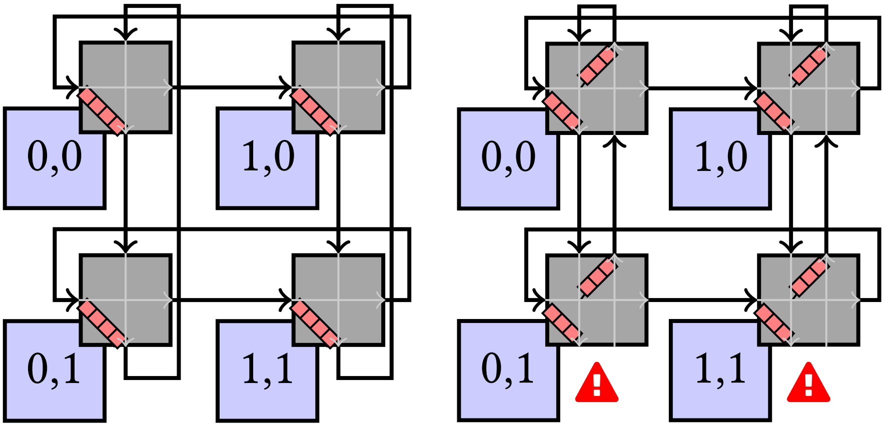
1 INTRODUCTION

> A well-regulated Network-on-Chip (NoC), being necessary to the safe operation of a real-time system, the right of the NoC packets to travel without unpredictable backpressure, shall not be infringed.
>
> —Ancient Klingon Proverb

High-performance communication networks are vital for supporting connectivity requirements of modern FPGA designs. FPGA logic can be configured to implement overlay packet-switched NoCs (or soft NoCs) [5, 7, 12, 14] to allow IP blocks to interact with each other at the cost of stealing logic and routing resources away from the developer. FPGA vendors are also embracing hardened NoC resources [13] that bake in the network functionality without using any soft logic. This trend is driven primarily by the communication demands of system-level IO. Soft NoCs still remain useful for two reasons: (1) for providing *last-mile* connectivity to portions of the FPGA fabric further away from hard NoC interfaces and (2) to deliver communication networks with greater flexibility and features such as timing guarantees that hard NoCs may not be able to provide.

Regardless of the choice of NoC implementation on FPGAs, real-time system developers wishing to use FPGAs need tool support to analyze the timing properties of their FPGA mappings to ensure they are able to meet relevant scheduling deadlines. Real-time systems are characterized by a need to rigorously prove timing requirements of various computing and communicating blocks. For instance, the ISO 26262 standard [6] requires performance isolation between communicating components on a chip. Shared communication media like NoCs are prone to interference and mixing of traffic that can complicate certification processes. NoC performance analysis is notoriously hard; it is often pessimistic and leads to overprovisioning of resources. In this article, we aim to build analysis-friendly, low-cost FPGA NoCs and develop accompanying analysis tools to prove worst-case NoC packet routing latencies.

Resource-efficient NoCs like Hoplite [7] and HopliteRT [14] provide a scalable, low-cost fabric for designing FPGA communication networks using soft logic. Both these NoCs are built on the idea of deflection routing that avoids the cost of packet buffering. The routers for these NoCs can be as small as 86–89 6-LUTs for 64-bit payloads operating at a 1.2–1.3ns clock period on a Virtex-7 485T FPGA. However, packets may suffer long deflection penalties in the fabric (Hoplite and HopliteRT), and packets may even suffer livelocks where packets deflect endlessly (Hoplite). This behavior is a problem for real-time FPGA applications in mission-critical environments like self-driving cars and automotive systems, unmanned drones, avionics, and biomedical devices. Such application domains need strict timing guarantees for bounding worst-case behavior and allowing certification of the products for use in the field. A conventional, buffered, packet-switched NoC might be a tempting alternative. However, deeply buffered NoCs with classic flow control are too expensive to implement on the FPGA and are hard to statically analyze to compute worst-case buffer bounds due to the complexity of packet interactions. Contemporary buffered FPGA NoCs like CMU CONNECT [12] and Penn Split-Merge [5] NoCs are very expensive and occupy thousands of LUTs/router for 32-bit routers (for context, a lightweight RISC-V CPU takes ≈ 200–500 LUTs on an FPGA [4]). The reason for such a bloated and slow design is that these routers support exotic features in a NoC, such as flow control, extensive buffering, arbitration logic, and virtual channels, which are not favorable for FPGA realization. Furthermore, the state-of-the-art analysis of NoC buffer bounds [8, 9] is pessimistic due to the complexity of modeling pipelining effects and mixing of various flows in the network. In this article, we propose the HopliteBuf NoC, shown in Figure 1, derived from the low-cost Hoplite NoC. HopliteBuf introduces (1) small stall-free buffers for certain router functions to simplify flow control to (2) eliminate deflections with any associated livelocks, and (3) provides optional linearization of vertical NoC rings to enhance analysis that



(a) $W \longrightarrow S$ Corner-Turn Buffer (b) $W \longrightarrow S + N$ Corner-Turn Buffer

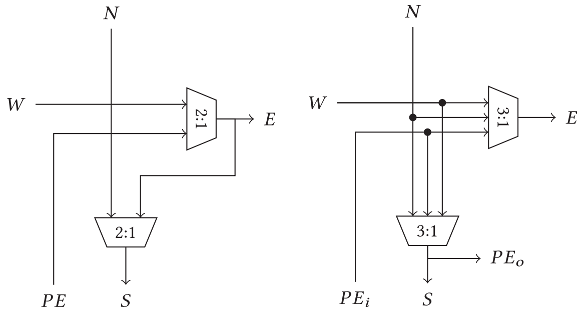
Fig. 1. 2×2 HopliteBuf NoC topology with stall-free corner-turn buffers (▨). Stall-free buffers are sized to never go full, avoiding the need for backward flow control. The $W \rightarrow S + N$ topology disconnects vertical rings (⚠) and introduces an extra uphill multiplexer in each router.

(4) bounds buffer sizes to distributed RAM-friendly implementation sizes. The key contributions of our work are the microarchitecture of analysis-friendly HopliteBuf NoC routers and topology modifications coupled with a buffer sizing algorithm that is able to determine the worst-case occupancies for all the FIFOs in the NoC. In particular, the proposed topology in Figure 1(b) with two corner-turn buffers and no vertical loopback simplifies static analysis of buffer sizes. This also improves provable wire utilization while preserving wiring cost and requiring a modest increase in LUT count over Figure 1(a). For our workloads we observe that these occupancies are small enough to be realizable in LUT-based FIFOs (Xilinx SRL32s and Intel MLABs). The HopliteBuf NoC is more expensive than Hoplite or HopliteRT by 3–4× due to buffering, but still cheaper than full-blown conventional buffered NoCs. We summarize the main contributions of our work here:

- Design of an FPGA NoC torus topology to enhance static analysis for computing buffer bounds and NoC router microarchitecture redesign with stall-free FIFOs to eliminate deflections and provide in-order packet delivery; optimization and customization of the NoC router RTL to match Xilinx and Intel FPGAs.
- Development of a buffer sizing algorithm to compute the worst-case bounds on FIFO occupancies; use of vertical NoC link linearization to improve provable link utilization. Static analysis tools compute upper bounds on size of FIFOs required for stall-free operation, source queueing delay, and in-flight routing latency under various conditions.
- Engineering of a robust simulation infrastructure to compute cycle counts of packet traversals in the NoC; resource and performance analysis of the NoC under various synthetic workloads.
- Development of RTL and analysis for Hoplite NoC router with backpressure in the horizontal rings. This allows us to quantify the cost advantages and the analysis benefits of using a HopliteBuf over conventional backpressure-based buffered routers.

2 BACKGROUND

We now review existing literature on deflection-routed FPGA overlay NoCs—Hoplite and HopliteRT—to highlight the underlying resource-performance tradeoffs, features, and limitations of these NoCs.



(a) Hoplite Router (b) HopliteRT Router

Fig. 2. High-level design sketches of Hoplite [7] (can livelock) and HopliteRT [14] (cannot livelock) FPGA NoC routers. HopliteRT adds a N→E turn to Hoplite to eliminate livelock. Both designs fit one bit of crossbar in one Xilinx 6-LUT.

Table 1. Routing Function Table to Support Single 6-LUT Implementation of Hoplite

Mux Select		Routes	Explanation
s_sel	e_sel		
0	0	PE→E or PE→S	No W or N packet
0	1	W→S or W→E	Even if PE has packet
1	0	N→S + PE→E	No W packet
1	1	N→S + W→E	No N packet

N has highest priority, followed by *W* port and *PE* has the least priority. *PE* and *W* cannot inject simultaneously if *W* has a packet even if the flows are nonoverlapping.

Hoplite [7] is an FPGA-friendly NoC router that uses torus topology and eliminates buffering and flow control to provide a low-cost implementation on modern FPGAs. Packets traverse using DOR (dimension-ordered routing) policy in the x-dimension (horizontally, W→E) first before turning (W→S) and routing in the y-dimension (vertically, N→S). While DOR is not strictly required for deflection-routed switches, Hoplite includes this routing scheme to reduce switching cost by eliminating certain turns in the router. This simple design requires a pair of 2:1 muxes as shown in Figure 2(a). The DOR logic defines the arbitration scheme with three inputs, N (North), W (West), and PE (processing element/client), and two outputs, S (South + Switch exit, shared) and E (East). The DOR control logic is very simple (shown in Table 1) and consumes very few LUTs as it can be constructed directly on valid signals of the incoming packets alone. Packets exiting the NoC must do so over the S port to allow the mux and wires to be shared by both southbound and exiting packets. A separate output valid signal helps determine the nature of the packet. The NoC client is provided the lowest priority and cannot inject a packet if the router has packets on both ports. A key limitation of Hoplite is the inability of the NoC to avoid livelock. This is possible as a W packet continues to get deflected to the E port as long as a packet on the N port wants to travel S. Furthermore, a series of packets sent from a source client to a destination client may take different paths through the network and need not deflect in an identical manner.

HopliteRT [14] is a refinement over Hoplite that inverts the priorities and deflects N packets to the E (hence the new N→E turn in Figure 2(b)). By doing this, HopliteRT achieves livelock freedom and bounds the worst-case latency as a function of system size. Thus, HopliteRT requires two 3:1 muxes but still requires the same number of LUTs as Hoplite as shown in Table 2. This is possible

Table 2. FPGA Costs for 64b Router (4 × 4 NoC) with Vivado 2016.4 (Default Settings) + Virtex-7 485T FPGA (Adapted from [14])

Router	LUTs	FFs	Period (ns)
Hoplite	89	149	1.29ns
HopliteRT	86	146	1.22ns

Table 3. Routing Function Table (Adapted from [14]) for HopliteRT

Mux select		Routes	Explanation
sel1	sel0		
0	0	W→E + N→S	Noninterfering
0	1	W→S + N→E	Conflict over S (not supported in Hoplite)
1	0	PE→E + N→S	No W packet
1	1	PE→S + W→E	No N packet (not possible in Hoplite)

PE injection has lowest priority and will stall on conflict. PE→E + W→S is not supported to avoid an extra select signal driving the multiplexers and doubling LUT cost by preventing fracturing a 6-LUT into 2 × 5-LUTs.

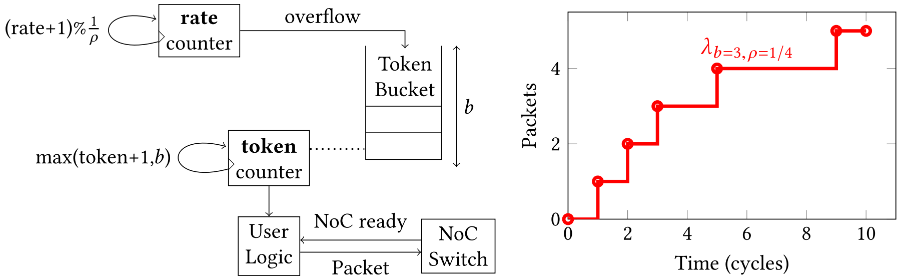
because the multiplexer inputs are shared and it requires a five-input, two-output function, which can be implemented using careful mapping using a LUT-6. HopliteRT overcomes the livelock limitation by forcing the N packet to deflect E in case of contention caused by high-priority W→ S flow. The N packet then reappears as a W packet with higher priority. This simple modification means that a packet will only suffer a single deflection at a given switch as it descends down the NoC. The adaptation not only avoids livelock but also puts an upper bound on in-flight NoC latency to $\Delta X + \Delta Y \times m + 2$ for an $m \times m$ NoC, where ΔX and ΔY, if the number of steps or nodes a packet has traversed in the X and Y direction, respectively. This indicates that, in the worst case (when $\Delta X = m$ and $\Delta Y = m$), the torus NoC could reduce down to a ring $O(m^2)$. We can see the complete routing function for HopliteRT in Table 3.

While HopliteRT achieves livelock freedom without increasing resource cost over Hoplite, it still has two drawbacks: (1) Due to the 1D ring linearization in the worst case, HopliteRT suffers from a high worst-case deflection bound. Reducing a bandwidth-rich torus to a 1D ring is neither an efficient nor scalable use of FPGA resources. (2) As the deflections are not ordered, packets can be delivered out of order at the destination. Reordering of packets now becomes the responsibility of the NoC client and it can add extra memory overheads at the endpoints.

The in-flight and source queueing latency bounds for Hoplite are ∞ due to the unpredictable nature of packet deflections. To bound *source queueing latency* (the amount of time a packet has to wait for a client before entering the network), HopliteRT uses the concept of traffic regulation.

Token Bucket Regulator: The objective of network regulation is to manage traffic congestion in a network of routers to allow guaranteed service to participating traffic flows. One such traffic control algorithm is Token Bucket regulation [11]. The regulator ensures that the data injected in the network conforms to defined limits on *rate* ρ and *burstiness* b. Here, *rate* is the throughput of packet injection measured in packets per second, while *burstiness* is the maximum number of back-to-back packets permitted to enter the network. These regulators have abundant applications in large-scale networks like internet routers.

We can customize these regulators for on-chip communication networks like FPGA NoCs. This regulator is highly efficient and cheap to implement, requiring just two counters. Figure 3(a) shows a high-level cartoon diagram of a Token Bucket Regulator inserted on the client → NoC interface.



(a) Token Bucket Regulator (b) Traffic Curve for regulator

Fig. 3. A cartoon diagram representing Token Bucket regulation and an example traffic curve with burst=3 and rate=$\frac{1}{4}$. A regulator is used at the injection port of each NoC client. Number of tokens represents the number of packets allowed to inject in the network by a client.

The regulator restricts the amount of traffic injected into the NoC and also bounds the amount of time a client has to wait to inject a packet into the network. Both ρ (injection rate) and b (burst) are configured by the NoC developer.

Two counters, a rate counter and a token counter, are required to implement Token Bucket regulation. A rate counter is a free-running counter that is programmed to add a token into the Token Bucket, shown in Figure 3(a), when it overflows after a period of every $\frac{1}{\rho}$ cycles. A token counter keeps track of tokens consumed by the client/user logic and makes sure that the Token Bucket never exceeds its maximum size, b. The Token Bucket is assumed to be initially full with b tokens (user programmable) in it. Whenever a client wishes to inject a packet in the network, it checks if the NoC is ready to accept packets and the client has enough tokens in the token counter. For each packet sent in the network, the token counter is decremented by one, which also represents that we have one empty slot in the Token Bucket.

We introduce the concept of a traffic curve $\lambda_{b,\rho}(t)$ to capture the maximum number of packets injected by a regulator in any interval of t cycles. Based on the provided description, it holds that

$$\lambda_{b,\rho}(t) = \min(t, b + \lfloor \rho \cdot (t-1) \rfloor). \tag{1}$$

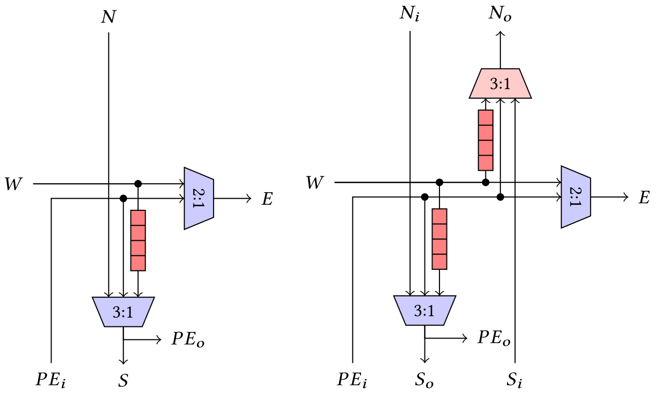
The traffic curve for a regulator with $b = 3, \rho = 1/4$ is depicted in Figure 3(b). The traffic curve derivation assumes that the bucket is initially full. Hence, $b = 3$ packets can be sent consecutively at times $t = 1, 2, 3$. After the first packet is sent at time $t = 1$, the regulator starts generating a new packet, which is then added to the bucket at time $1 + 1/\rho = 5$; this corresponds to the fourth transmitted packet. Afterward, new packets are sent every $1/\rho$ cycles.

3 HOPLITEBUF MICROARCHITECTURE

In this section, we describe the design of the HopliteBuf router with stall-free buffering. We explain the core switch microarchitecture and associated routing policy and discuss FPGA mapping.

3.1 The Idea

Earlier in Figure 1, we sketched two variants of the proposed HopliteBuf topologies with buffers for turning packets. In Figure 4, we show the switch microarchitectures of these variants. The basic multiplexing functionality implements turns to support the DOR routing scheme. What this means is that, unlike the HopliteRT design, our proposed microarchitecture does not support the $N \rightarrow E$ turn. Now, recall that contention in Hoplite arises from packets wanting the same S resource either for turns ($W \rightarrow S$) or for vertical descent ($N \rightarrow S$). We can choose to make either or both of these conflicting parties wait in buffers, but the $W \rightarrow S$ option is preferred as it limits buffering



(a) FIFO on $W \rightarrow S$ Turn (b) FIFOs on $W \rightarrow S + N$ Turns

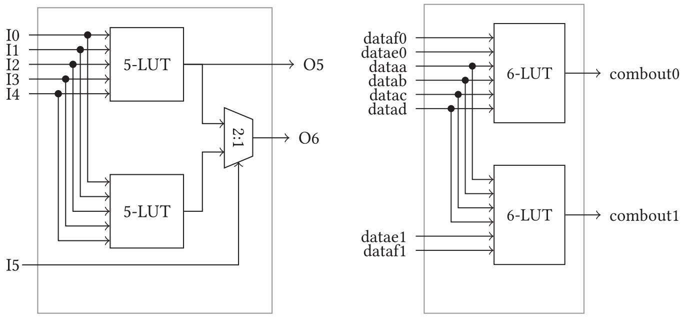
Fig. 4. Two design alternatives for adding buffers to the Hoplite NoC router. $W \rightarrow S$ adds a buffer on the corner turn, while $W \rightarrow S + N$ adds an extra uphill buffer.

penalty for a given flow to a **single** buffer. Buffering the $N \rightarrow S$ path will force packets descending vertically to wait at each hop, prolonging their stay in the network. This would make both end-to-end routing latency and FIFO size larger than needed. Hence, we focus only on W packets for buffering. We now elaborate on the two design options that only buffer W packets in two ways:

- ***W*** **→** ***S*** **buffer:** In this scenario, packets turning from the W port to S will be buffered if a $N \rightarrow S$ packet arrives at that router in the same cycle. The routing policy now prioritizes N packets over W packets as there is no longer the option of deflecting to E like in HopliteRT. The East mux now sees W and PE as input, and the South mux sees W' (FIFO output), N, and PE as inputs. The multiplexer select lines also need to be distinct as the routing combinations prevent sharing. We discuss how this may fit on the FPGA fracturable LUT organization in Section 3.2 and the restrictions of the routing combinations in Section 4. From the perspective of the NoC, the packet will have to wait in a buffer **only** at the point of turn. The PE_o exit shares the same wires as the S, just like in the original Hoplite and HopliteRT routers, to avoid paying the extra cost of exit multiplexers. Empirical evaluation has shown a negligible performance hit from this cost-saving transformation.
- ***W*** **→** ***S*** **+** ***N*** **buffer:** In this second scheme, the routing policy introduces a $S \rightarrow N$ link and allows a new $W \rightarrow N$ turn. At first glance, this may seem like an unlikely design choice as inserting an entirely new routing path will increase LUT resource costs. While this is true, this scheme does **not** increase wiring requirements, as seen in Figure 1(b). The vertical wrap-around link in the original Hoplite ring is now forced to traverse through the switch on the way uphill. Thus, total wirelength stays unchanged. Furthermore, as we will see later, this organization enhances the static analysis pass by removing the loopback and allows a higher provable link utilization on the vertical ring. You may notice we retain the shared single exit to the client PE_o that shares wires with the S port. As the vertical ring is disconnected, traffic is delivered to destination PEs only on the downhill traversal. This is another cost-saving measure that eliminates introducing an exit multiplexer along with an accompanying FIFO for packets on the uphill $S_i \rightarrow N_o$ that may wish to exit sooner.

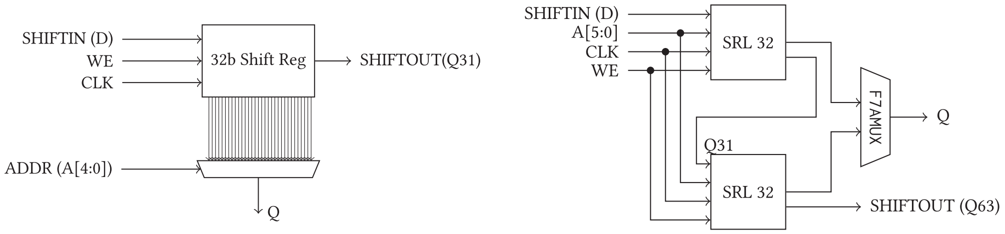
3.2 FPGA Implementation

We show the distinct LUT organizations of the Xilinx and Intel FPGAs in Figure 5. A Xilinx 6-LUT is fracturable into two 5-LUTs (shown in Figure 5(a)) with five common inputs across both LUTs.



(a) Xilinx Lookup Table (LUT) Organization (b) Intel Adaptive Logic Module (ALM) Organization

Fig. 5. Xilinx and Intel FPGA logic organizations. A fracturable Xilinx six-input Lookup Table (LUT) can fit two 5-LUTs with common inputs. A fracturable Intel Adaptive Logic Module (ALM) can fit a variety of Adaptive LUT (ALUT) combinations, with two 6-ALUTs with common inputs combination shown.



(a) Xilinx 32b Shift Register (b) Xilinx 64b Shift Register

Fig. 6. (a) A LUT-6-based 32-bit Shift Register on Xilinx FPGA. (b) Two LUT-6 cascaded to implement a 64-bit Shift Register. The cascading is done with the help of F7AMUX, which is available in SLICEM of a Combinational Logic Block in Xilinx FPGAs.

This allows you to implement one function of five-inputs (any function) and one function of six inputs (if it overlaps with the five-input function) in the same LUT. An Intel ALUT is organized differently (shown in Figure 5(b)) and has eight inputs shared across two 6-LUTs. The two 6-LUTs have four common inputs, and two distinct inputs each. They can implement two functions of six inputs as long as they share four inputs.

The Xilinx LUT structure is extremely flexible and can be configured as Storage elements, such as Distributed RAM and Shift Register [16]. Figure 6(a) shows how a LUT can be configured to work as a 32-bit SRL. The 32:1 mux selects one of the 32 inputs coming from the Shift Register based on the address port A. Adding a slight amount of logic on top of this module can make it work like a 32-deep 1-bit FIFO. Multiple LUTs can be cascaded to increase the *depth* and *width* of the FIFO implemented using SRLs. One such example is shown in Figure 6(b), where two LUTs are cascaded such that the FIFO depth is 64. In a similar manner, within a SLICE, a maximum of four 32-bit SRL32 can be cascaded with the help of F7AMUX and F8AMUX to implement a 128-bit Shift Register or FIFO. A detailed discussion of realizing FIFOs on Xilinx SRLs is provided in [3]. This is integrated into our routers with a minor modification that the FIFO *full* signal is guaranteed never to go high based on static analysis and remains unused in our logic.

Similar to what Xilinx LUTs can do to implement Shift Register and memories, Intel implements this functionality with their MLAB resources. MLABs are a superset of LAB and support all the features that LABs can support. MLABs have a slightly different architecture to build the memory

Table 4. Resource Utilization on Xilinx Virtex-7 and Intel Arria-10 Devices for Different Data Widths and FIFO Sizes of HopliteRT and HopliteBuf Routers

	Xilinx						**Intel**					
	HopliteRT		**W → S**		**W → S+N**		**HopliteRT**		**W → S**		**W → S+N**	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
DW=32, FIFO=32	59	86	155	94	-	-	102	92	167	91	-	-
DW=64, FIFO=32	91	150	251	158	-	-	166	156	263	144	-	-
DW=32, FIFO=64	59	86	197	95	262	142	102	92	248	140	278	156
DW=64, FIFO=64	91	150	325	159	413	238	166	156	408	221	438	246
DW=32, FIFO=128	59	86	281	96	346	144	102	92	331	224	445	261
DW=64, FIFO=128	91	150	482	160	562	240	166	156	555	360	733	409

The LUTs used for FIFO storage are computed as per the equation Datawidth $\cdot \lceil \frac{FIFO}{32} \rceil$.

configuration. Each MLAB supports a maximum of 640 bits of simple dual-port SRAM. MLABs can be configured to implement 32-deep and 20-bit-wide simple dual-port SRAMs. For more information, please refer to [1]. The stall-free buffer component of our new designs is implemented using these cheap resources.

Original Hoplite Mapping: When implementing the original Hoplite router shown in Figure 2(a) on a Xilinx FPGA, we can easily fit the two 2:1 muxes in two Xilinx 5-LUTs to allow a compact one 6-LUT mapping per bit of switching. This is possible as the East 2:1 mux can use a 5-LUT (requiring three inputs), while the South 2:1 mux can be mapped to the embedded 2:1 mux that drives the O6 output (needing two more inputs, only one of which needs to be unique). When implementing the original Hoplite router on an Intel FPGA, it is trivially possible to fit this in a single ALM with two 6-LUTs without even forcing the East mux serialization. This is because the 3:1 mux (South mux if implemented fully) needs five inputs, while the 2:1 mux only needs three. Two of these inputs are shared by both muxes (W and PE_i), while the distinct mux-select inputs can be supplied to the two 6-LUTs independently without violating the common input restriction.

Mapping $W \rightarrow S$ FIFO Design: This design requires the switching crossbar to consume four packet inputs, N, W, W' (FIFO output), and PE_i, along with three mux-select inputs. This already exceeds the 6-input LUT capacity of the Xilinx FPGA and cannot use fracturing. On the Intel FPGA, we require four common inputs to each 6-LUT; this constraint is satisfied by our design, thereby enabling a compact fit. Additionally, we can supply two unique inputs to each 6-LUT, which is adequate to support the mux-select signals. When it comes to implementing FIFOs using this cheap resource, both Xilinx and Intel FPGAs provide configuration to use LUT and ALMs as Memory registers. To implement deeper FIFOs with Xilinx, we use the cascading strategy shown in Figure 6(b), where multiple LUTs can be cascaded with each other by using `F7AMUX` and `F8AMUX`. With Intel devices, deeper memories can be built much easily by just instantiating multiple MLAB primitives explained in [1].

Mapping $W \rightarrow S + N$ FIFO Design: This design requires the switching crossbar to consume six packet inputs, N_i, S_i, W, W' ($W \rightarrow S$ FIFO), W'' ($W \rightarrow N$ FIFO), and PE_i, and five mux-select inputs. We choose to split the turning packets into separate FIFOs to prevent mutual interference between traversing flows. The total distributed RAM capacity stays the same as it just split into two SRL32 or MLAB instantiations instead of a longer single distributed RAM block. Here, with limited opportunity for input sharing, the resulting design is larger in LUT cost, but as we will see later, this allows efficient use of the NoC links. In this design as well, we implement the memory components using LUTs/MLABs similar to the $W \rightarrow S$ buffer design.

Table 4 shows the resource costs of implementing different variants of HopliteBuf on Xilinx and Intel FPGAs. The maximum FIFO depth analyzed in this work is capped at 128 to reasonably implement them using LUTs and ALMs. As shown in the table, the W → S design is slightly more expensive than HopliteRT on Xilinx FPGAs due to their LUT architecture. However, the W → S design uses the same number of ALMs to implement the logic as HopliteRT but is a little expensive in terms of buffer implementation using MLABs. In the dual-buffer variant, W → S+N, the memory costs are identical to W → S, but it is slightly more expensive due to an added 3:1 mux to implement the upward logic. On average, HopliteBuf is around 3–4× more expensive than HopliteRT, but as you will see in the evaluation section, this added cost is going to pay in terms of performance gain over HopliteRT.

4 ROUTING POLICY

Now that we understand the architecture and implementation details of the two variants of HopliteBuf, $W \rightarrow S$ and $W \rightarrow S + N$, we are going to look at the arbitration scheme and routing policies to implement these structures.

The original Hoplite and HopliteRT routers implemented bufferless deflection routing rooted in the Dimension-Ordered Routing policy. The policy ensured that arriving packets from W and N ports were sent to E and S ports, respectively. For turning packets, Hoplite prioritizes the N port over the W port, thereby introducing the possibility of livelock, while HopliteRT prioritizes W over N to ensure bounded NoC routing delays. Thus, HopliteRT deviates from DOR by allowing a $N \rightarrow E$ deflection that is not permitted under standard DOR implementation.

HopliteBuf: $W \rightarrow S$ Design: For $W \rightarrow S$ design, we restore the DOR routing policy as we move back to the same architecture (with an added buffer on the W to S link) as Hoplite. But in order to service FIFO packets, the routing policy has been modified. We list the routing combinations for the $W \rightarrow S$ NoC design in Table 9 in the appendix.

The upper half of the routing table shows the possible flow combinations when only one port, either W, W', N, or PE_i, has packets to transmit. The lower half shows more complicated routing decision cases when multiple ports are willing to transmit simultaneously. For South mux, the arbitration scheme gives the highest priority to the N port, as it is not buffered and cannot hold the packets. The second-highest priority is given to W' (West FIFO output), while PE_i gets the least priority. For East mux, the W port has higher priority than PE_i packets. The decision logic also takes care of a unique and complex routing case of simultaneous data transfer between $W \rightarrow E$ and $W' \rightarrow S$.

HopliteBuf: $W \rightarrow S + N$ design: For the $W \rightarrow S + N$ design as well, we use the DOR routing policy. With buffering, W packets are forced to wait in the FIFOs, thereby transferring priority to N_i and S_i packets. This design still accepts PE_i packets with the least priority. We list the routing combinations for the $W \rightarrow S + N$ NoC design in Table 10 in the appendix. The routing logic is much more complex in comparison to the $W \rightarrow S$ design as it has eight possible flow combinations and all have different priority structures. The table summarizes all the possible combinations along with their priority.

5 W → S BUFFER DESIGN WITH BACKPRESSURE

In contrast to HopliteBuf, where FIFOs are provably stall free, conventional buffered NoCs use backpressure signaling to propagate the FIFO stall condition upstream. This backpressure signal propagates upstream through potentially several routers, stalling each link along the way. This architecture allows analysis-free operation of the NoC but also suffers from poorer latency outcomes due to complex network flow interactions. To quantify the benefits of HopliteBuf over this conventional alternative, we develop the W→S + Backpressure design. We develop RTL for this design as well as network flow-based analysis. In particular, for the torus topology used by our work, this design resembles the Kim NoC Router [10].

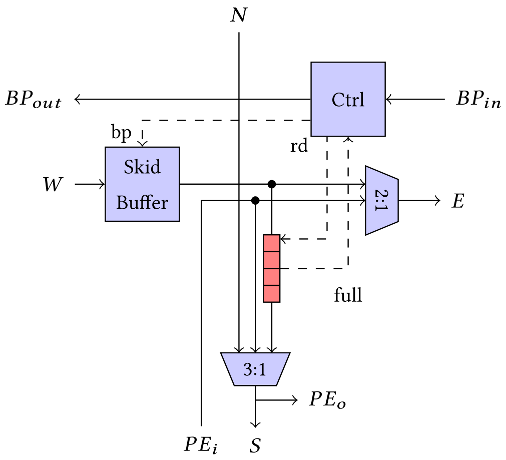


Fig. 7. FIFO on $W \rightarrow S$ turn along with Backpressure unit.

The microarchitecture of this router introduces two primary hardware structures into the design as shown in Figure 7:

- A *Backpressure* controller is added to manage the propagation or generation of backpressure indication. Backpressure may be *generated* when a turning W→S packet encounters a full W→S FIFO. Alternatively, backpressure may be *propagated* along the W→E link if an eastbound packet encounters a stalled upstream West port of a router.
- A shadow register or skid buffer is needed on the W port of the router to hold packets in place as the backpressure signal propagates horizontally in a pipelined fashion. The effect of pipelining introduces a one-cycle delay on the backpressure signal it traverses through each router. Packets may have begun their traversals before they saw the incoming backpressure. This is handled using a standard shadow register or skid buffer on the W input.

In this design, the vertical N→S packets continue to enjoy backpressure-free traversal in the NoC. Backpressure only affects the horizontal packets and may inadvertently hold up an W→E packet if a downstream W→S FIFO has gone full. This blocking behavior is not possible in HopliteBuf, as all packets in front of a packet along the horizontal link are guaranteed safe storage in the turn FIFOs should they need to do so. The DOR routing logic must be adapted to account for the presence of backpressure and FIFO full signals. In Table 11 in the appendix, we show the complete routing table used in our design.

Another impact of this microarchitecture is the larger expected cost. The backpressure controller and the shadow register will increase LUT and FF usage, while also slowing down clock frequency due to complex control flow. We quantify the result of hardware mapping of the routers on Xilinx and Intel FPGAs in Table 5. Finally, note that for the purpose of network-flow based analysis, the W→S + Backpressure design will consider turn FIFO sizes to be only one deep. Deeper FIFOs require queuing analysis and generate pessimistic bounds due to increased flow burstiness, as we will explain in Section 6.

6 LATENCY AND BUFFER SIZE ANALYSIS

We now turn our attention to static analysis of the NoC traffic to bound buffer sizes and worst-case injection (source queueing) and in-flight traversal latencies. This is important to establish whether we can realize these buffers in distributed FPGA RAMs (SRLs and MLABs). We first introduce

Table 5. Resource Utilization on Xilinx Virtex-7 and Intel Arria-10 Devices for Different Datawidths and FIFO Sizes for $W \rightarrow S_{bkp}$ Design

	Xilinx			**Intel**		
	LUTs	FFs	%	LUTs	FFs	%
DW=32, FIFO=32	246	175	58.7	257	159	53.8
DW=64, FIFO=32	409	303	62.9	416	227	58.1
DW=32, FIFO=64	290	176	47.2	338	184	36.2
DW=64, FIFO=64	485	304	49.2	561	304	37.5
DW=32, FIFO=128	372	177	32.3	420	247	26.8
DW=64, FIFO=128	640	305	32.7	707	450	27.3

% increase over $W \rightarrow S$ switch is also quantified.

our regulation and traffic model. We then develop a network calculus approach to FIFO size and worst-case latency analysis for HopliteBuf. The presence of cycles in the torus topology makes this analysis susceptible to instability, but we are able to provide an analytic solution that employs a topology linearization alternative (Figure 1(b)) to eliminate cycles and get accurate buffer bounds and latencies.

6.1 Traffic and Network Model

Injection regulation is a known technique to establish well-defined behavior of network traffic at runtime for off-chip internet-scale systems. As discussed in Section 2, we adapt Token Bucket regulation at the NoC clients to enforce traffic discipline on the NoC. This is done transparently and the datapath design just needs to obey the standard NoC valid-ready interface (AXI-stream). We can implement this regulation on the FPGA using two simple counters per NoC client and require **no** buffers at the client-NoC interface.

We consider an $(m \times m)$ matrix of clients (x, y). Each client sends packets as part of one or more flows; all packets within the same flow have the same destination and use the same token bucket regulator. Hence, we use $F = \{f_1, \ldots, f_i, \ldots\}$ to denote the set of flows in the system, where for each flow f: $(f.xs, f.ys)$ represents the source client of the flow; $(f.xd, f.yd)$ represents the destination client; and $f.b, f.\rho$ represent the regulator parameters. Note that two different flows f_i and f_j might share the same source, or the same destination.

Example: We present a running example of an NoC with five flows $f_{1\ldots5}$ using the $W \rightarrow S$ buffer design in Figure 8. Note that we use f_i' to denote a flow after it leaves a buffer, as buffering can increase the burstiness of the flow (packets queued up in a buffer can be flushed directly back to back). Relevant flow parameters are tabulated in Table 6.

The amount of traffic carried by each flow is represented by a traffic curve $\lambda_{b,\rho}(t)$, as detailed in Equation (1), where b and ρ are the parameters of the corresponding regulator. Flows entering from different ports into a switch might affect the traffic injection rate by the client at that router. To determine the effect, we need to analyze the combined traffic rate of all flows entering the router. To calculate the combined traffic load, we consider two traffic flows, λ_{b_1,ρ_1} and λ_{b_2,ρ_2}, entering a router from two different ports and directed to the same output. Lemma 1 defines an operator $\oplus$ that combines the two traffic curves to compute a tight bound on the resulting aggregated traffic.

Lemma 1 (Lemma 1 in [15]). *Let λ_{b_1,ρ_1} and λ_{b_2,ρ_2} bound the traffic on two input ports (West, North, or PE) directed to the same output port (East or South). Then the traffic on the output port is bounded*

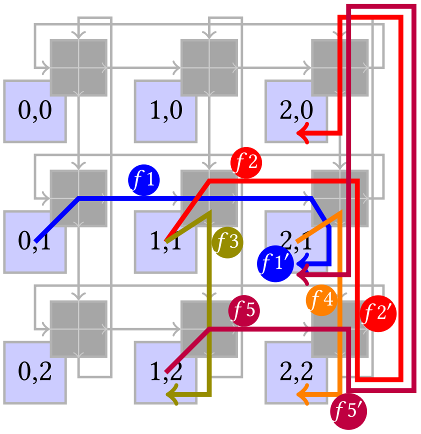


Fig. 8. Example $W \to S$ NoC design with five flows $f_{1\dots5}$.

Table 6. Flow Parameters for the Example NoC

Flow	Source	Dest	Γ^C	$f_{W\to S}$	$f_{N\to S}$
f_1	(0,1)	(2,1)	none	f_2	f_5'
f_2	(1,1)	(2,0)	f_3, f_1	f_1	f_5'
f_3	(1,1)	(1,2)	f_2	-	-
f_4	(2,1)	(2,2)	f_1', f_2', f_5'	-	-
f_5	(1,2)	(2,1)	none	none	$f_2' + f_4$

Γ^C are the conflicting flows used in Section 6.2; $f_{W\to S}$ and $f_{N\to S}$ are the $W \to S$ and $N \to S$ interfering flows used in Section 6.3. '-' denotes not applicable.

by the following curve:

$$(\lambda_{b_1,\rho_1} \oplus \lambda_{b_2,\rho_2})(t) = \min\left(t, b_1 + b_2 + \lfloor \rho_1 \cdot (t-1) \rfloor + \lfloor \rho_2 \cdot (t-1) \rfloor\right). \tag{2}$$

More in general, let $\mathcal{A}$ be a set of traffic curves representing traffic directed to the same port, and then the traffic on that port is bounded by

$$\oplus\mathcal{A}(t) = \min\left(t, \sum_{\forall \lambda_{b,\rho} \in \mathcal{A}} b + \lfloor \rho \cdot (t-1) \rfloor\right). \tag{3}$$

Our analysis derives three sets of parameters:

- Injection latency $Injection(f)$ for each flow $f \in F$; this is the maximum time that the source client $(f.xs, f.ys)$ can be stalled waiting to send a packet of f.
- Maximum queuing delay $Delay(f)$ for each flow f turning $W \to S$ through a buffer.
- Backlog for each router; this is the maximum number of packets that are queued waiting to be transmitted (excluding the packet that might be transmitted in the current clock cycle).

The total latency for a flow f is then obtained as the sum of $Injection(f)$, $Delay(f)$ and the total per-hop delay of one cycle per hop, which is equal to $(f.yd - f.ys)\%m + (f.xd - f.xs)\%m + 1$.

We discuss how to compute the injection latency in Section 6.2. We then show how to derive delay and backlog bounds for the $W \to S$ design in Section 6.3, and for the $W \to S + N$ design in Section 6.5. Note that the analysis for $W \to S$ is harder, due to the loopback of the vertical ring. The instability created by loopbacks is a notoriously challenging problem [11] and results in lower provable bounds on link utilization. The $W \rightarrow S + N$ design does not suffer such issue as the flows have been linearized and have no loops. Finally, we extend the $W \rightarrow S$ analysis to handle backpressure in Section 6.6.

6.2 Injection Latency

We now show how to compute the maximum delay suffered by a client (x, y) to inject a sequence of k packets of flow f, where $k \leq f.b$.

- The first step is to determine the set of *conflicting flows* Γ^C, that is, those flows that block the injection of packets at the analyzed client. It comprises all other flows injected by the same source client, since a client can inject only one packet per cycle. For the example shown in Figure 8, Table 6 shows that there are no conflicting flows for flow f_1 since no other flows are originating from source (0,1). However, f_3 becomes a conflicting flow for flow f_2 since f_3 is originating from the same source at f_2.
- For the second step, we need to add to Γ^C all the flows generated by other clients that traverse the same mux used by f at its source router $(f.xs, fxs)$. If f injects packets to the East port, then it suffers conflicts from any flow $W \rightarrow E$. If f injects packets to the South port, then it suffers conflicts from flows $W \rightarrow S$ or $N \rightarrow S$. For the example shown in Figure 8, Table 6 shows that f_2 is injecting packets to the East port and the East traversing f_1 is conflicting with it. For f_4, which is injecting to the South port, the interfering flows consist of f_1' and f_2', the flows that turn South on that router after traversing the corresponding West buffer, and f_5', which traverses the router $N \rightarrow S$.

Assume that each flow in Γ^C is bounded by a traffic curve $\lambda_{b,\rho}(t)$; we define $b(\Gamma^C)$ as the sum of burstiness parameters b of traffic curves for all flows in Γ^C, and $\rho(\Gamma^C)$ as the sum of their rate parameters. To compute the upper bound on packet injection latency, we can employ Lemmas 2 and 3 in [15], which prove that under the condition $\rho(\Gamma^C) < 1$, the number of cycles where a client is free to inject on the NoC in any interval of t cycles is bound by

$$t - \oplus\Gamma^C(t) \geq \max\left(0, \lfloor\left(t - (T^s + 1)\right) \cdot \left(1 - \rho(\Gamma^C)\right)\rfloor + 1\right), \tag{4}$$

where

$$T^s = \left\lceil \frac{b(\Gamma^C)}{1 - \rho(\Gamma^C)} \right\rceil. \tag{5}$$

Note that this implies that the flow might receive no free cycles for T^s clock cycles but is then guaranteed to receive slots at a rate of $1 - \rho(\Gamma^C)$. Based on Lemmas 2 and 3 in [15], the following theorem can be proven.

Theorem 1 (Theorem 2 in [15]). *Assume $\rho(\Gamma^C) < 1$ and the client wishes to inject a sequence of $k \leq f.b$ packets for flow f. Then the delay to inject all packets in the sequence is upper bounded by*

$$\boxed{Injection(f, k) = \lceil 1/f.\rho \rceil - 1 + T^s + \left\lceil (k-1) \cdot \max\left(\frac{1}{f.\rho}, \frac{1}{1 - \rho(\Gamma^C)}\right) \right\rceil}. \tag{6}$$

Based on Theorem 1, the first packet in the sequence waits for at most $\lceil 1/f.\rho \rceil - 1 + T^s$ cycles; successive packets are sent either every $1/f.\rho$ or every $1/(1 - \rho(\Gamma^C))$ cycles, whichever is higher. Note that this implies that if $f.\rho + \rho(\Gamma^C) > 1$, meaning that the NoC is saturated, then the node cannot inject at its prescribed rate $f.\rho$, and is instead forced to inject at a lower rate $1 - \rho(\Gamma^C)$. For this reason in our evaluation, if the condition $f.\rho + \rho(\Gamma^C) \leq 1$ is not satisfied, we consider the

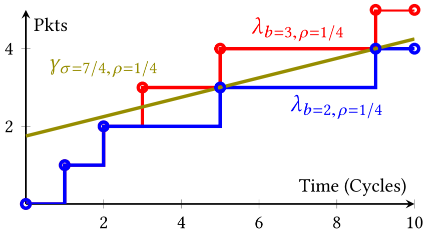


Fig. 9. Example traffic curves for $\lambda_{b=3,\rho=1/4}$ and $\lambda_{b=2,\rho=1/4}$ along with an arrival curve for $\gamma_{b=7/4,\rho=1/4}$ (discrete version).

flow set unfeasible. It remains to determine the traffic curve $\lambda_{b,\rho}(t)$ for each interfering flow. For a flow f_i that has not yet traversed a buffer, the curve is simply $\lambda_{f.b,f.\rho}(t)$. We show how to derive the traffic curve for a flow f_i' that leaves a $W \rightarrow S$ buffer in the next section.

6.3 Vertical Ring Analysis $W \rightarrow S$ Design

We now analyze the behaviour of flows turning on a vertical ring through a $W \rightarrow S$ buffer. We employ the theory of network calculus [11] for FIFO-arbitrated flows to derive deterministic bounds on the queuing delay and backlog. In particular, we show that the delay and backlog depend on the burstiness and rate of flows entering the FIFO buffer, as well as the burstiness and rate of flows routed $N \rightarrow S$. The theory describes the behavior of dataflows traversing a system of network elements. Each element is composed of a buffer, where incoming data is stored, and a server, which takes data from the buffer and forwards it. Each dataflow is represented by an arrival curve $\alpha(t)$, which bounds the maximum amount of data in any interval of length t. A commonly used curve is the *leaky bucket curve* $\gamma_{\sigma,\rho}(t)$, which is defined as

$$\gamma_{\sigma,\rho}(t) = \sigma + \rho \cdot t. \tag{7}$$

Whenever a flow f is bounded by a curve $\gamma_{\sigma,\rho}(t)$, we shall also use $f.\sigma$ and $f.\rho$ to denote the arrival curve parameters for the flow.

We can convert between the traffic curve representation $\lambda_{b,\rho}(t)$, which we used to bound the injection latency, and arrival curves $\gamma_{\sigma,\rho}(t)$ based on the following lemma:

Lemma 2. *(1) A flow bounded by traffic curve $\lambda_{b,\rho}(t)$ is also bounded by arrival curve $\gamma_{b-\rho,\rho}(t)$. (2) Similarly, a flow bounded by arrival curve $\gamma_{\sigma,\rho}(t)$ on any NoC link is also bounded by traffic curve $\lambda_{\lceil\sigma+\rho+1\rceil,\rho}(t)$.*

Proof. Part (1). Based on the curve definitions, we have

$$\begin{aligned}\lambda_{b,\rho}(t) &= \min(t, b + \lfloor \rho \cdot (t-1) \rfloor)\\ &\leq b + \rho \cdot (t-1) = b - \rho + \rho \cdot t = \gamma_{b-\rho,\rho}(t).\end{aligned}$$

Part (2). Again, by definition:

$$\begin{aligned}\gamma_{\sigma,\rho}(t) &= \sigma + \rho \cdot t = \sigma + \rho + \rho \cdot (t-1) \leq \sigma + \rho + \lfloor \rho \cdot (t-1) \rfloor + 1\\ &\leq \lceil \sigma + \rho + 1 \rceil + \lfloor \rho \cdot (t-1) \rfloor.\end{aligned}$$

Since furthermore an NoC link cannot transmit more than one packet every clock cycle, the flow is bounded by

$$\min(t, \lceil \sigma + \rho + 1 \rceil + \lfloor \rho \cdot (t-1) \rfloor) = \lambda_{\lceil\sigma+\rho+1\rceil,\rho}(t). \qquad \square$$

Example: Figure 9 shows traffic curves for two regulators with $b = 2, \rho = 1/4$ and $b = 3, \rho = 1/4$. The traffic curve $\lambda_{b=2,\rho=1/4}(t)$ is upper bounded by $\gamma_{\sigma=7/4,\rho=1/4}(t)$. Similarly, arrival

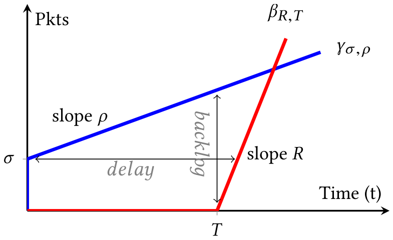


Fig. 10. Arrival ($\gamma_{\sigma,\rho}$) and service ($\beta_{R,T}$) curve example.

curve $\gamma_{\sigma=7/4,\rho=1/4}(t)$ is upper bounded by $\lambda_{b=\lceil 7/4+1/4+1\rceil,\rho=1/4}(t) = \lambda_{b=3,\rho=1/4}(t)$; $\gamma_{\sigma=7/4,\rho=1/4}(t) > \lambda_{b=3,\rho=1/4}(t)$ for $t = 1, 2$, but since the NoC link cannot send more than one packet per cycle, $\lambda_{b=3,\rho=1/4}(t)$ is still a valid traffic bound. In essence, Lemma 2 allows us to "convert" a flow with a traffic curve $\lambda_{b,\rho}(t)$ into an arrival curve $\gamma_{\sigma,\rho}(t)$ and vice versa, albeit at some loss of precision.

There are situations where we need to aggregate (combine) flows transmitted on the same link. Note that for two arrival curves $\gamma_{\sigma',\rho'}(t)$ and $\gamma_{\sigma'',\rho''}(t)$, it immediately holds that $\gamma_{\sigma',\rho'}(t) + \gamma_{\sigma'',\rho''}(t) = \gamma_{\sigma'+\sigma'',\rho'+\rho''}(t)$: hence, the arrival curve for the aggregate of flows traversing the same link can be expressed by summing the σ and ρ parameters of the arrival curves for the individual flows.

We next consider the behavior of the server. We model it through a *strict service curve* $\beta(t)$, which is a lower bound to the amount of data forwarded by the element in any interval of length t under the condition that it is backlogged during the interval; that is, the buffer is never empty. A commonly used service curve is the *rate-latency curve* $\beta_{R,T}(t)$, which is defined as

$$\beta_{R,T}(t) = \max(0, R \cdot (t - T)). \tag{8}$$

Given the arrival curve and service curve, one can compute bounds on delay and backlog as follows:

Theorem 2 (Theorems 1.4.1 and 1.4.2 in [11]). *Consider a network element with service curve $\beta_{R,T}(t)$ traversed by a flow with arrival curve $\gamma_{\sigma,\rho}(t)$. If $\rho < R$, then*

$$Delay = T + \sigma/R \tag{9}$$

$$Backlog = \sigma + \rho \cdot T. \tag{10}$$

A graphical interpretation of Theorem 2 is shown in Figure 10; note that the delay is the maximum horizontal distance between the arrival and service curve, while the backlog is the maximum vertical distance.

Figure 11 illustrates the flows required for analysis at one $W \rightarrow S$ NoC router. Here, f and f' represent a flow under analysis before and after leaving the $W \rightarrow S$ buffer; $f_{W\rightarrow S}$ represents the aggregate of all other interfering flows traversing the buffer; $f_{N\rightarrow S}$ represents the aggregate of all interfering flows traversing the router in the $N \rightarrow S$ direction; and $f_{PE\rightarrow S}$ represents the aggregate of all flows injected by the client at that router directly S. As discussed in Section 4, the S mux arbitration gives lowest priority to the client; hence, we do not have to consider flow $f_{PE\rightarrow S}$ when analyzing flow f, but it will interfere in the $N \rightarrow S$ direction on the next router. Regarding the other flows, $f_{N\rightarrow S}$ has higher priority than f, while $f_{W\rightarrow S}$ and f are FIFO scheduled as they traverse the same FIFO buffer.

Assuming that each flow is described by an arrival curve, we will use two further lemmas from [11] that will helps us analyze the system.

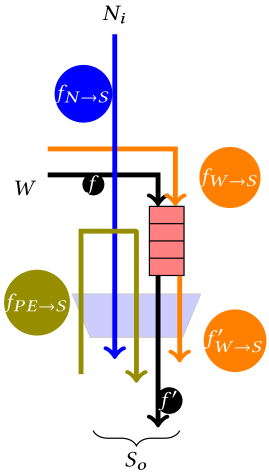


Fig. 11. Flows through a $W \to S$ router.

Lemma 3 (Proposition 1.3.4 in [11]). *Consider a network element that forwards data at a constant rate C with no processing delay. Assume that the element services two flows f_H and f_L, where f_H has higher priority than f_L. Further assume that f_H is bounded by a leaky bucket arrival curve with $f_H.\rho < C$. Then f_L is guaranteed a service curve $\beta_{R,T}$ with $R = C - f_H.\rho$ and $T = \frac{f_H.\sigma}{C - f_H.\rho}$.*

Based on Lemma 3, we can generate a service curve $\beta_{R,T}$ for the aggregate of flows f and $f_{W\to S}$, where f_H is $f_{N\to S}$. Since the router transmits one packet per cycle, we have $C = 1$, thus yielding

$$R = 1 - f_{N\to S}.\rho, \tag{11}$$

$$T = \frac{f_{N\to S}.\sigma}{1 - f_{N\to S}.\rho}. \tag{12}$$

We can then compute the backlog by applying Theorem 2 to the aggregate of f and $f_{W\to S}$:

$$\boxed{Backlog = f.\sigma + f_{W\to S}.\sigma + (f.\rho + f_{W\to S}.\rho) \cdot \frac{f_{N\to S}.\sigma}{1 - f_{N\to S}.\rho}}. \tag{13}$$

Lemma 4 (Corollary 6.2.3 in [11]). *Consider a network element serving two flows f_1, f_2 bounded by leaky bucket arrival curves in FIFO order. Assume that the element guarantees to the aggregate of the two flows a service curve $\beta_{R,T}$, and that furthermore $f_1.\rho + f_2.\rho < R$. Then f_1 has a service curve $\beta_{R',T'}$ with $R' = R - f_2.\rho$ and $T' = T + \frac{f_2.\sigma}{R}$. Furthermore, the output flow f_1' is constrained by a leaky bucket arrival curve with $f_1'.\rho = f_1.\rho$ and $f_1'.\sigma = f_1.\sigma + f_1.\rho(T + \frac{f_2.\sigma}{R})$.*

By applying Lemma 4 to our router, where f_1 is the flow under analysis, f_2 is $f_{W\to S}$, and R, T are obtained in Equations (11) and (12), we obtain service curve $\beta_{R',T'}$ for f:

$$R' = 1 - f_{N\to S}.\rho - f_{W\to S}.\rho, \tag{14}$$

$$T' = \frac{f_{N\to S}.\sigma + f_{W\to S}.\sigma}{1 - f_{N\to S}.\rho}. \tag{15}$$

Substituting the values of T' and R' in Equation (9), we obtain

$$\boxed{Delay(f) = \frac{f.\sigma}{1 - f_{N\to S}.\rho - f_{W\to S}.\rho} + \frac{f_{N\to S}.\sigma + f_{W\to S}.\sigma}{1 - f_{N\to S}.\rho}} \tag{16}$$

under the condition that $f.\rho + f_{N\to S}.\rho + f_{W\to S}.\rho < 1$ (i.e., the link is not saturated).

Finally, note that unless it immediately exits the network, flow f' will be injected on the North port of the next router. Hence, the $f_{N\to S}$ flow for the next router can include f'. For this reason, we have to compute an arrival curve for f' to be able to analyze such a router. Again, applying Lemma 4, we obtain

$$\boxed{\begin{aligned} &f'.\rho = f.\rho \\ &f'.\sigma = f.\sigma + f.\rho \cdot \frac{f_{N\to S}.\sigma + f_{W\to S}.\sigma}{1 - f_{N\to S}.\rho} \end{aligned}} \; . \qquad \begin{matrix}(17)\\[2ex](18)\end{matrix}$$

under the nonsaturation condition that $f.\rho + f_{N\to S}.\rho + f_{W\to S}.\rho < 1$. Since Equations (17) and (18) must hold for all flows f_i turning $W \to S$ on the vertical ring, we effectively obtain a system of equations. Once the system is solved, the flow $N \to S$ on each router can be determined, and Equations (13) and (16) can be used to obtain backlog bounds for all buffers and queuing delays for all flows. Based on Equation (17), buffering does not increase the rate of flows. Furthermore, based on Lemma 2, for any flow f_i that has not been buffered, we have $f_i.\sigma = f_i.b - f_i.\rho$. Hence, the only unknowns in Equation (18) are the values $f_i'.\sigma$ for flows that have crossed a buffer. To analyze the system, we thus apply the so-called Time Stopping Method in network calculus [11]: we treat the values $f_i'.\sigma$ as variables, obtaining a system of linear equations. If the values of $f_i'.\sigma$ obtained by solving the system of equations are valid (i.e., bounded and positive), then $\gamma_{f_i'.\sigma, f_i'.\rho}(t)$ upper bounds flow f_i'. Otherwise, the network cannot be analyzed.

Example: Let us consider flows at router (2,1) from Figure 8. Assume $f_1.\rho + f_2.\rho + f_5.\rho < 1$. For flow f_1, $f_{W->S}$ consists of flow f_2, while $f_{N->S}$ consists of flow f_5'. Since for any flow $f_i.\sigma = f_i'.\sigma$ and $f_i.\sigma = f_i.b - f_i.\rho$, we obtain

$$f_1'.\sigma = f_1.b - f_1.\rho + f_1.\rho \cdot (f_5'.\sigma + f_2.b - f_2.\rho)/(1 - f_5.\rho).$$

Similarly, applying Equation (18) to flows f_2, f_5 under the added assumption $f_2.\rho + f_4.\rho + f_5.\rho < 1$ yields

$$\begin{aligned} f_2'.\sigma &= f_2.b - f_2.\rho + f_2.\rho \cdot (f_5'.\sigma + f_1.b - f_1.\rho)/(1 - f_5.\rho), \\ f_5'.\sigma &= f_5.b - f_5.\rho + f_5.\rho \cdot (f_2'.\sigma + f_4.b - f_4.\rho)/(1 - f_2.\rho - f_4.\rho). \end{aligned}$$

Hence, we solve a linear system of three equations to determine the value of variables $f_1'.\sigma, f_2'.\sigma, f_5'.\sigma$, which can then be used to determine the backlog at each router and delay for each flow according to Equations (13) and (16). Furthermore, by applying Lemma 2, we derive equivalent traffic curves $\lambda_{\lceil f_i'.\sigma + f_i'.\rho + 1\rceil, f_i'.\rho}(t)$ for f_1', f_2', and f_5', which we use to bound the injection latency of f_4. As an example, if we set $b = 1$, $\rho = 1/4$ for all regulators, we obtain $f_1'.\sigma = f_2'.\sigma = 33/20$ and $f_5'.\sigma = 39/20$, which result in backlogs of 14/5 at (2, 1) and 39/20 at (2, 2). Hence, we need a minimum $W \to S$ buffer size of $\lfloor 14/5 \rfloor + 1 = 3$ at (2, 1) and $\lfloor 39/20 \rfloor + 1 = 2$ at (2, 2); note that we add 1 to the buffer size to account for a packet being read from the buffer and transmitted in the current clock cycle.

While the presented approach generally results in reasonably tight bounds, it is known [2, 11] that the circular dependencies introduced by a ring design can reduce the sustainable (provable)

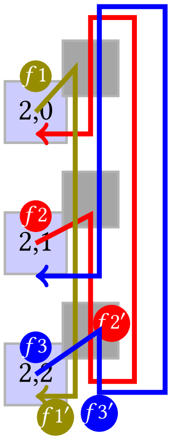


Fig. 12. Cyclic dependency flow example in $W \rightarrow S$ HopliteBuf.

per-link utilization of the network by up to 50%. We present an example of circular dependencies and show how to handle them in the next section.

6.4 Cyclic Dependencies in HopliteBuf $W \rightarrow S$ Design

Figure 12 shows example flows for a vertical ring, where each flow enters in a router and exits from the router above. Assuming that $f_1.\rho + f_2.\rho + f_3.\rho < 1$ so that no link is saturated, we can use Equation (18) to write a system of three linear equations:

$$f_1'.\sigma = f_1.\sigma + f_1.\rho \cdot \frac{f_2'.\sigma + f_3'.\sigma}{1 - f_2.\rho - f_3.\rho} \tag{19}$$

$$f_2'.\sigma = f_2.\sigma + f_2.\rho \cdot \frac{f_1'.\sigma + f_3'.\sigma}{1 - f_1.\rho - f_3.\rho} \tag{20}$$

$$f_3'.\sigma = f_3.\sigma + f_3.\rho \cdot \frac{f_1'.\sigma + f_2'.\sigma}{1 - f_1.\rho - f_2.\rho}. \tag{21}$$

Let us now assume that $f_1.\rho = f_2.\rho = f_3.\rho = \rho < \frac{1}{3}$. We can rewrite the system of equations as follows:

$$\vec{\sigma} = A \cdot \vec{\sigma} + \vec{a}, \vec{\sigma} = \begin{bmatrix} f_1'.\sigma \\ f_2'.\sigma \\ f_3'.\sigma \end{bmatrix}, A = \begin{bmatrix} 0 & \frac{\rho}{1-2\rho} & \frac{\rho}{1-2\rho} \\ \frac{\rho}{1-2\rho} & 0 & \frac{\rho}{1-2\rho} \\ \frac{\rho}{1-2\rho} & \frac{\rho}{1-2\rho} & 0 \end{bmatrix}, \vec{a} = \begin{bmatrix} f_1.\sigma \\ f_2.\sigma \\ f_3.\sigma \end{bmatrix}, \tag{22}$$

where A is a positive matrix and $\vec{a}$ a positive vector of constant terms. Assuming that the spectral radius of A is less than 1, the matrix $(I - A)$ is invertible; we can thus obtain the unknown vector $\vec{\sigma}$ as $\vec{\sigma} = (I - A)^{-1} \cdot \vec{a}$. Defining $\eta = \frac{\rho}{1-2\rho}$ and using some linear algebra, we obtain

$$(I - A)^{-1} = \begin{bmatrix} \frac{1-\eta^2}{1-3\eta^2-2\eta^3} & \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} & \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} \\ \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} & \frac{1-\eta^2}{1-3\eta^2-2\eta^3} & \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} \\ \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} & \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} & \frac{1-\eta^2}{1-3\eta^2-2\eta^3} \end{bmatrix}. \tag{23}$$

It is now possible to prove [11] that the spectral radius condition is equivalent to $(I - A)^{-1}$ being positive, which yields the condition $0 \leq \eta < \frac{1}{2}$, or equivalently $0 \leq \rho < 0.25$. This means that the

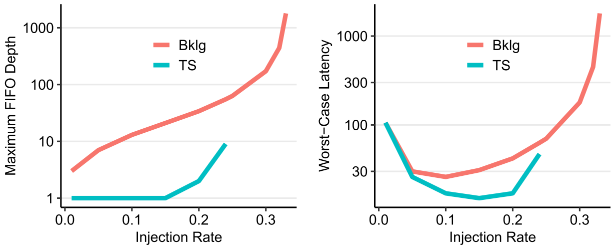


Fig. 13. Comparing Backlog-based (Bklg) bound calculation with Time-stopping (TS) method when computing FIFO size and worst-case latency. While the Backlog-based bound permits scaling to higher injection rates, the bounds are significantly worse. In practice, we find that when limited by FIFO capacity, the Backlog-based bound generally becomes infeasible and the gains from supporting higher rates are lost.

maximum per-link utilization on the network is $3 \cdot 0.25 = 75\%$, lower than the 100% utilization implied by condition $f_1.\rho + f_3.\rho + f_3.\rho < 1$.

To address such issue, the authors of [11] also demonstrate a different approach, known as the Backlog-based Method.

Theorem 3 (Theorem 6.4.1 in [11]). *Consider a unidirectional ring of m network elements that forward data of a flow aggregate at constant rate C with no processing delay. Let $F^1, \ldots, F^m$ denote the set of flows on each of the m network elements. Further define:*

$$\sigma_{\text{tot}} = \sum_{f \in F^1 \cup \ldots \cup F^m} f.\sigma, \tag{24}$$

$$\sigma_{\max} = \max_{i=1\ldots m} \sum_{f \in F^i} f.\sigma, \tag{25}$$

$$\rho_{\max} = \max_{i=1\ldots m} \sum_{f \in F^i} f.\rho. \tag{26}$$

Then if $\rho_{\max} < C$, the backlog at any router is bounded by

$$Backlog = m^2 \cdot \frac{\rho_{\max}}{C - \rho_{\max}} \cdot \sigma_{\max} + \sigma_{\text{tot}}, \tag{27}$$

and the maximum queuing delay at any network element is $Backlog/C$.

We can apply Theorem 3 to our system by setting $C = 1$ and computing the F^i set based on the flows traversing the *ith* router in either the $N \rightarrow S$ or $W \rightarrow S$ direction; as before, we do not consider the flows $PE \rightarrow S$ as they do not affect buffering at the router where they are injected, but they must be considered in the $N \rightarrow S$ set at the router below.

Since Theorem 3 always produces a bound as long as $\rho_{max} < 1$, the Backlog-based method allows us to analyze networks with up to 100% per-link utilization. However, as we show in Figure 13, in practice the obtained bound is highly pessimistic. It is clear that the peak injection rate supported by Backlog-based analysis is as high as 33% per flow, which yields a 99% link utilization. This is better than the 75% link utilization possible with the Time-stopping method. However, the pessimism in the Backlog-based analysis shows as much as 5–20× larger FIFO capacities and up to 3× worse base latency than the Time-stopping method. Furthermore, when applied to larger workloads, the wins do not materialize and Time-stopping remains the better approach (see our detailed evaluation in Section 7.3.4). For this reason, we next discuss the $W \rightarrow S + N$ design, which leads to much tighter bounds by linearizing the vertical communication.

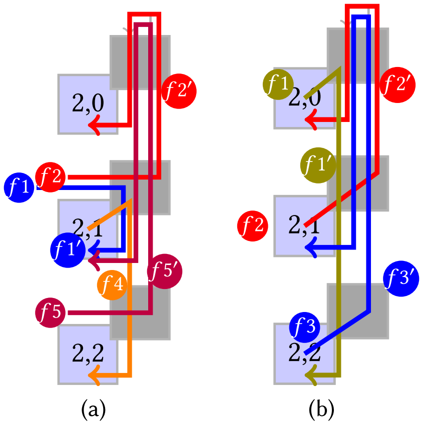


Fig. 14. (a) $W \rightarrow S + N$ design example: rightmost column. (b) Breaking cyclic dependency with linearization in $W \rightarrow S + N$ design example.

Table 7. Conflicting and Interfering Flows for the $W \rightarrow S + N$ Design

Flow	Γ^C	$f_{W\rightarrow S}$	$f_{N\rightarrow S}$	$f_{W\rightarrow N}$	$f_{S\rightarrow N}$
f_1	none	none	f_5'	-	-
f_2	f_1, f_3	-	-	none	f_5'
f_4	f_1', f_5'	-	-	-	-
f_5	none	-	-	none	none

'-' denotes not applicable, as the flow is not buffered in that direction.

6.5 Linearized Analysis: $W \rightarrow S + N$ Design

The analysis for the $W \rightarrow S + N$ design proceeds in a similar manner but is much simpler as no vertical loopback exists. The same injection latency computation is performed, although the set Γ^C can be different compared to the $W \rightarrow S$ design since a flow that was conflicting on the S mux could now turn N instead. Similarly, the same conditions in Equations (13), (16), (17), and (18) can be applied after decoupling each router into two parts: a south component containing the $W \rightarrow S$ buffer and S mux, and a north component containing the $W \rightarrow N$ buffer and N mux. Since packets are transmitted in different directions for the two components, when writing the equation for the north components, we use flows $f_{S\rightarrow N}$ and $f_{W\rightarrow N}$ in place of $f_{N\rightarrow S}$ and $f_{W\rightarrow S}$.

Example: Figure 14(a) shows the resulting decomposition for the rightmost column of the flow set depicted in Figure 8. Note that the topmost router $(2, 0)$ only implements the South component, as no flow can be injected North at $(2, 0)$. The sets of conflicting flows Γ^C and interfering flows $f_{N\rightarrow S}, f_{W\rightarrow S}, f_{S\rightarrow N}, f_{W\rightarrow N}$ are provided in Table 7. Compared to the $W \rightarrow S$ design, the number of conflicting and interfering flows is reduced.

When compared to the $W \rightarrow S$ design, we do not need to solve a system of equations to compute the $f_i'.\sigma$ values: since the $W \rightarrow S + N$ design disconnects vertical rings, we can apply Equation (18) to flows with destinations on a column x by ordering the flows based on the router at which they turn, in the order of packet propagation: from $(x, m-1)$ to $(x, 1)$ for flows turning North, and then from $(x, 0)$ back to $(x, m-1)$ for flows turning South. As long as no link is saturated, it is guaranteed that the analysis will compute bounded delay and backlog.

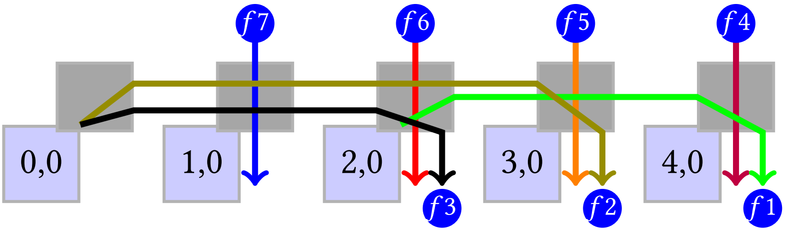


Fig. 15. Example backpressure scenario on horizontal ring.

Table 8. Generated and Propagated Backpressure Sets for the Example in Figure 15

Router	Generated Set	Propagated Set
$(4,0)$	f_4	-
$(3,0)$	f_5	f_4
$(2,0)$	f_6	f_4, f_5
$(1,0)$	-	f_4, f_5, f_6

6.6 Backpressure Analysis for $W \to S$ Design

Finally, we discuss how to modify the analysis to handle the backpressure design in Section 5. Consider again the flows in Figure 11, but assume that the router implements backpressure. As explained in Section 5, in this case the routers do not queue packets. Hence, the maximum number of packets of flow f that are forwarded South is bounded by the number of packets of f arriving from the West; and flow f' is bounded by the same arrival curve as f, meaning it holds that $f'.\sigma = f.\sigma$ in addition to $f'.\rho = f.\rho$. Therefore, backpressure reduces the burstiness of the output flow f' compared to the $W \to S$ buffer design. In addition, f does not suffer any queuing delay. However, f can suffer backpressure every time a packet of a flow in $f_{N \to S}$ traverses the router. Hence, the flows in $f_{N \to S}$ must be added to the set of conflicting flows Γ^C, which increases the injection latency for f. In this case, we say that f suffers *generated backpressure* from flows in $f_{N \to S}$.

We also have to consider the effect of *propagated backpressure* on each horizontal link. Assume that two flows f_1 and f_2 share a West port on a router west of $f_1.xd$. Then any packet that causes backpressure on f_1 can also cuase backpressure on f_2. Hence, all flows that cause generated backpressure on f_1 also cause propagated backpressure f_2. If another flow f_3 exists such that f_2 and f_3 share a West port on the router further west, flows that cause either generated or propagated backpressure on f_2 also cause propagated backpressure on f_3, and so on. In practice, we can construct a set of backpressuring flows at each router in a horizontal link by first deriving the set of generated backpressure flows and then propagating such flows backward through the horizontal ring as long as there are flows sharing the same West port. Once the set of backpressuring flows at each router is computed, it can be added to the Γ^C set for each flow traversing the West port at that router.

Example: Consider the example depicted in Figure 15 and Table 8, showing a horizontal ring with three flows $f_{1\ldots3}$ where all routers use backpressure. We start by computing the generated backpressure flows for the West port of each router, which are f_4 for $(4,0)$, f_5 for $(3,0)$, and f_6 for $(2,0)$ (note that f_7 does not cause backpressure, since no flow is turning $W \to S$ at $(1,0)$). Since there is a flow that crosses the West port at both $(4,0)$ and the previous router $(3,0)$ (flow f_1),

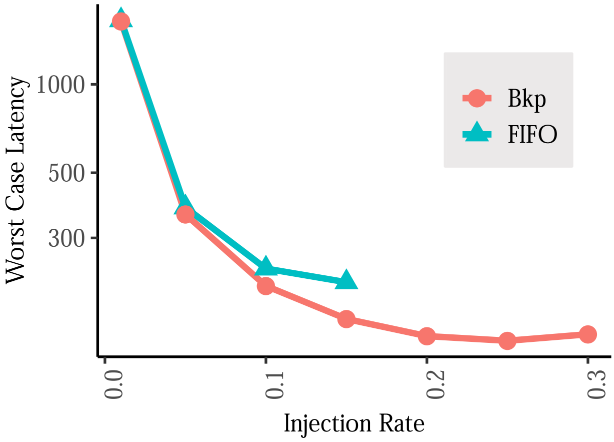


Fig. 16. Worst-case latency trends comparing W→S stall-free and backpressure designs on the example flows in Figure 12. All three flows have a burst of 16. The W→S stall-free analysis uses the Time-stopping method.

we add the backpressuring flows at (4, 0) to the set of propagated backpressure at (3, 0). Similarly, since there is a flow that crosses the West port at both (3, 0) and (2, 0), (1, 0) (flow f_2), we add the backpressuring flows at (3, 0) to the set of propagated ones at (2, 0) and (1, 0), and the ones at (2, 0) to the ones at (1, 0). Finally, the backpressure sets are added to the conflicting set Γ^C based on the West ports traversed by each flow; here, f_4 and f_5 are added to the conflicting set for f_1, while flows f_4, f_5, and f_6 are added to the conflicting sets for f_2 and f_3. Finally, note that while in this example the propagation stops at (1, 0), in general due to the torus topology, a $N \rightarrow S$ flow might have to be added to either the generated or propagated set for all routers on a given horizontal ring.

The discussed example shows that dependencies between horizontal flows can significantly grow the set of conflicting flows for f. Hence, injection latency can be significantly worse for the backpressure case compared to the previous designs. However, since the backpressure architecture does not queue packets, it does not suffer from the issue of cyclic dependencies for vertical flows. Specifically, consider again the three-flows example in Figure 12; as pointed out in Section 6.4, the $W \rightarrow S$ design supports a maximum per-link utilization of 45% based on the Time-stopping method. In contrast, as shown in Figure 16, the backpressure design supports a maximum per-link utilization of 99% (33% injection rate for each flow). Furthermore, we observe lower worst-case latency. In summary, this indicates a tradeoff between the two designs, where results depend on the topology of the analyzed flow set; in our evaluation in Section 7.3, we provide an in-depth analytical comparison based on random flows.

7 EVALUATION

We present the performance measurement results for our FPGA-optimized NoC and associated results from static analysis. We are interested in understanding the worst-case NoC routing latency properties, its breakdown, buffer depth bounds, and routing coverage. We also want to confirm the properties of static analysis bounds and understand their impact of distributed FPGA RAM mapping costs. We show results for 5 × 5 NoCs to retain narrative consistency but can generate other RTL networks and bounds for other sizes as well. We study four synthetic workloads that are commonly used in the real-time systems community:

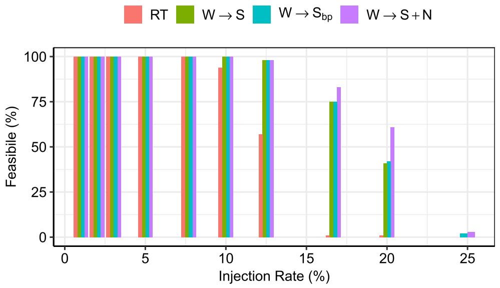


Fig. 17. Feasible flowsets for RANDOM traffic with $b = 1$ at 5×5 system size with 128-deep FIFOs in the NoC routers.

- We use the ALL-TO-ONE pattern that gets all NoC clients to target the same NoC address (destination PE (0,0)): a shared resource like an external DRAM, PCIe, or Network port.
- We use the ALL-TO-ROW pattern that gets all NoC clients to target the same row in the torus (destination row 0).
- We use the ALL-TO-COLUMN pattern that gets all NoC clients to target the same column in the torus (destination column 0).
- We also use the synthetic uniform RANDOM traffic pattern that is expressed as a set of flows, i.e., flowsets. We evaluate the NoCs using 100 separately generated synthetic flowsets. Each flowset is a collection of m^2 distinct streaming flows. Each flow captures data communication between a source-destination pair of clients. All flows have the same rate, which is increased until the links saturate. For simplicity, we also assign the same burstiness $b = 1$ to all flows, but also consider larger bursts in later analysis.

7.1 RTL Simulation Results

We first examine the results (feasibility, latency, FIFO sizing) of cycle-accurate RTL simulations of the different NoCs.

7.1.1 Flowset Feasibility. For our designs, we cap the maximum FIFO occupancy at 128 to enable low-cost realizations. As a result, some combination of flowset communication pattern and injection rate ρ will likely be infeasible. If any FIFO ever goes full, we classify that configuration as not feasible. We want to know what fraction of our 100 randomly generated flowsets were able to route without any of the NoC FIFOs ever going full at a given rate. In Figure 17, we plot the number of feasible flowsets for the RANDOM traffic pattern on the different NoCs. For HopliteRT, there are no FIFOs, but we know that flowsets are not feasible when the interfering flows on any link exceed the link bandwidth; i.e., you cannot use more than 100% of any link capacity. For the W → S buffer design with Backpressuring ($W \rightarrow S_{bp}$), FIFO occupancy can never exceed the maximum programmed depth (128 in this case) and hence a flow in this design becomes infeasible when the link bandwidth exceeds 100%.

The deflection pattern for HopliteRT forces traffic to travel through longer paths through the NoC, thereby interfering with a lot of other traffic flows. Hence, the feasibility trends for HopliteRT fall drastically above 10% injection rates. The HopliteBuf NoCs (W → S, W → S+N) are more resilient and support a larger fraction of the flowsets for larger injection rates. At the peak supported injection rate of 20%, HopliteBuf supports up to 50–60% of the flowsets, while HopliteRT only routes 1–2% of the flowsets. As predicted from the linearization analysis in Section 6.5, the $W \rightarrow S + N$ topology allows the system to support more traffic and a slightly greater fraction of the synthetic combinations are feasible at even 20% injection rates. The backpressure design,

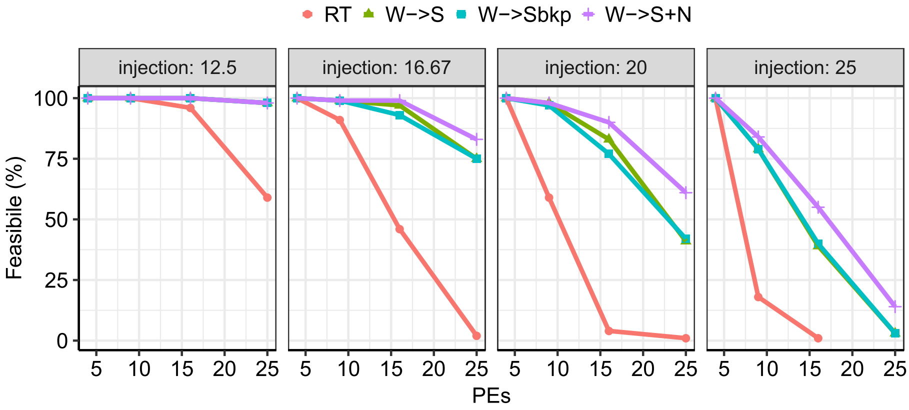


Fig. 18. Feasible flowsets for RANDOM traffic with $b = 1$, FIFO depth = 128 with different system sizes and injection rates.

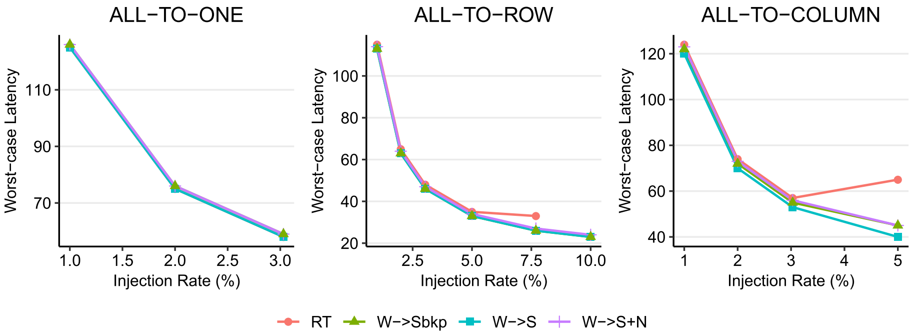


Fig. 19. Worst-case latency trends for ALL-TO-ONE, ALL-TO-ROW, and ALL-TO-COLUMN traffic patterns on NoCs with 5 × 5 system sizes and $b = 1$. HopliteBuf designs offer no improvements for these traffic patterns.

$W \rightarrow S_{bp}$, performs identically to the HopliteBuf $W \rightarrow S$ design; however, at 20% we see a slightly better performance with this design. But the dual buffer design $W \rightarrow S + N$ still outperforms all the other designs. At 25%, we observe that no other design is feasible except for $W \rightarrow S_{bp}$ and $W \rightarrow S + N$, where $W \rightarrow S + N$ is still outperforming the backpressure design. Higher feasibility translates into more FPGA developer freedom in being able to support their communication requirements.

In Figure 18, we plot the number of feasible flowsets for the RANDOM traffic pattern on the different system sizes. With increasing injection rates, HopliteRT becomes less feasible for all system sizes. For smaller systems, HopliteRT performs identically to buffered NoC designs; however, for large systems, HopliteRT suffers a drastic reduction in feasibility at higher injection rates.

7.1.2 Worst-Case Latency Trends. We expect that the use of buffering will help reduce worst-case routing latencies as we eliminate deflections. However, the improvements will be balanced by the penalty of waiting in the FIFOs. In Figure 19 we show this effect for three traffic patterns with burst $b = 1$, and in Figure 20 we show the same effect for 100 RANDOM flowsets. The common odd trend here is the *decrease* in injection latency as a function of injection rate. This is not an illusion and is a result of the fact that the client is regulated and may miss the token cycle that

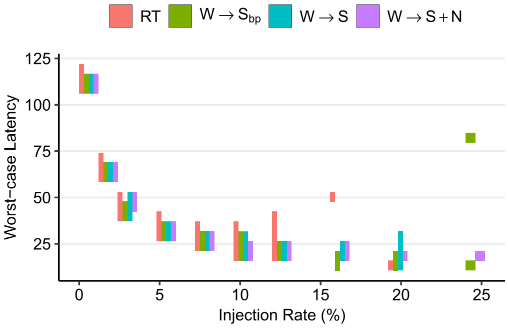


Fig. 20. Worst-case latency trends for RANDOM traffic pattern on NoCs with 5 × 5 system sizes and $b = 1$. HopliteBuf performs better than other designs for this workload.

scales with the injection rate ρ of the regulator. At large enough injection rates we eventually start to see an increase due to network congestion, but this is marginal. For the ALL-TO-ONE traffic pattern, the waiting time in the FIFOs lines up with the penalty of deflections, resulting in no observable difference between the different designs. For the ALL-TO-ROW traffic pattern, there is no visible difference among the buffered NoC designs except that HopliteRT saturates at a 7.5% injection rate while buffered NoCs are saturating at a higher injection rate of 10%. For ALL-TO-COLUMN, all the NoC designs are saturating at around 5% and they all perform almost identically except for HopliteRT, which shows substantially higher worst-case latency than other designs. The saturation rates for these designs are in agreement with the system size, where all the other nodes send packets randomly to each other except to itself. For RANDOM traffic, we show a distribution of measured cycle counts across the 100 flowsets. There is a clear benefit to using buffers to avoid deflections as bufferless HopliteRT shows a wider spread of achieved worst-case latencies. The buffer waiting time is lower than the penalty of deflections, resulting in tighter latency spreads for HopliteBuf NoCs. Furthermore, $W \rightarrow S$ designs suffer a buffer wait only at a single turn; it exhibits slightly better or identical performance to the two-FIFO $W \rightarrow S + N$ design. The backpressure design $W \rightarrow S_{bp}$, however, performs similarly to the HopliteBuf $W \rightarrow S$ variant and in some case even better than the dual-buffer $W \rightarrow S + N$ design. Overall, HopliteBuf is 1.2–2× better than HopliteRT in terms of worse-case routing latencies. We also see that HopliteRT is poorly unable to support the highest injection rate of 20% that is well supported by the HopliteBuf NoCs. Thus, the presence of buffers not only improves (reduces) worst-case latencies but also supports higher data rates. This is expected as HopliteRT and $W \rightarrow S_{bp}$ steal unnecessary bandwidth in the X-ring due to deflection or backpressuring.

7.1.3 Worst-Case Latency Breakdown. In Figure 21 we show a breakdown of worst-case latency into its source-queueing latency (waiting time at PEs) and in-flight latency (actual routing time in the NoC). The improvements due to elimination of deflections do show up in better in-flight routing latencies for HopliteBuf designs, but larger wins are visible during source queueing. This is because the NoC is blocking the PE injection ports less often by keeping packets in the buffers instead of wasting injection slots due to deflection.

For the HopliteRT routing scheme, the $N \rightarrow E$ deflection potentially sends packets along the *scenic route* around each X-ring (at most once), generating traffic conflicts where none would exist for conventional DOR routing. The $W \rightarrow S_{bp}$ design has comparable in-flight latencies, but we expect the source-queuing latency for this design to go high in case of small buffer sizes as *PE* will be backpressured and packets will wait longer in the client if the buffer is full. HopliteBuf

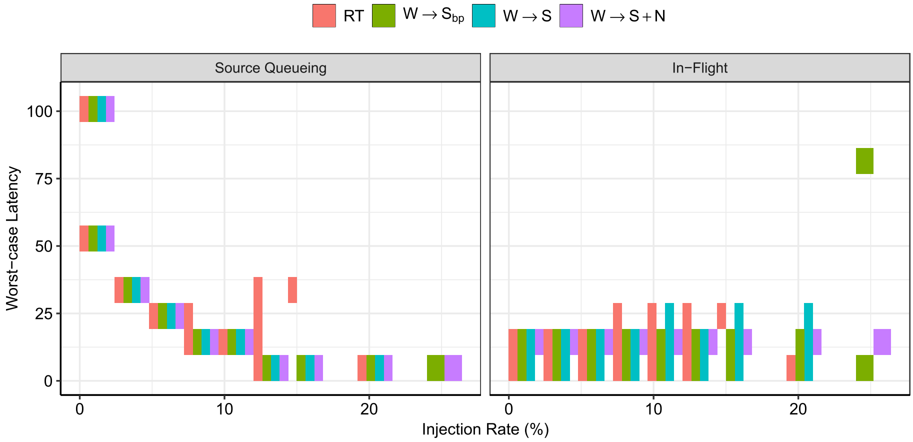


Fig. 21. Breakdown of source-queueing and in-flight NoC latencies for RANDOM workload with $b = 1$, FIFO depth = 128 at 5×5 system size. Both metrics improved due to buffering.

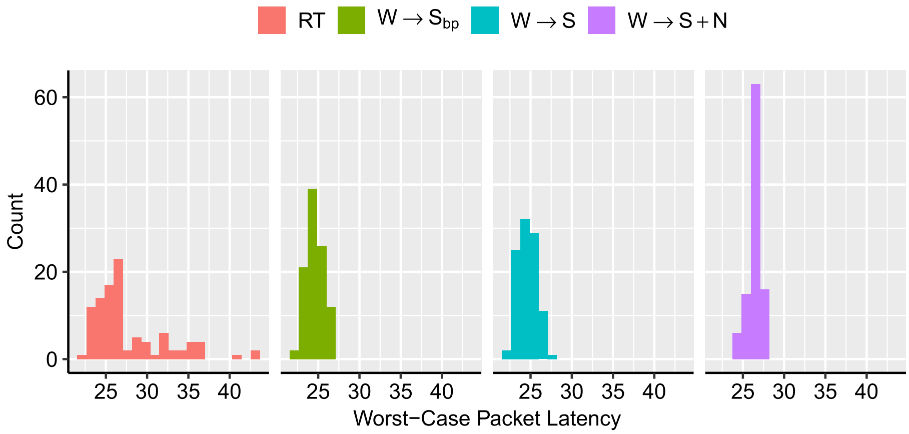


Fig. 22. Distribution of worst-case packet latencies for RANDOM workload with $b = 1$, $\rho = 7.5\%$ at 5×5 system size. HopliteRT has a wider spread due to the unpredictable nature of the deflections. HopliteBuf has narrower spreads.

chooses FIFO waiting on conflicts, thereby reducing contention in other X-rings and a drop in source queueing delays. As we see, the NoC traversal time is mostly unaffected even in the presence of FIFOs.

7.1.4 Latency Distribution. In Figure 22, we show the histogram of worst-case packet latencies for the different NoCs for RANDOM traffic with burst $b = 1$ and injection rate $\rho = 7.5\%$ at 5×5 system size. We note that the HopliteRT NoC has a much wider spread than the FIFO designs. This is because deflections create unpredictable trips through the NoC X-rings. In contrast, a victimized packet just sits in a buffer and the waiting time in the buffer is much lower than round trips around the ring. We see that the $W \rightarrow S_{bp}$ design performs better than the Hoplite $W \rightarrow S$ design as more cases are feasible with backpressuring than with just buffers. As expected, all the other buffered designs have a marginally wider distribution than the $W \rightarrow S + N$ design as the packets have an extra choice during the turn.

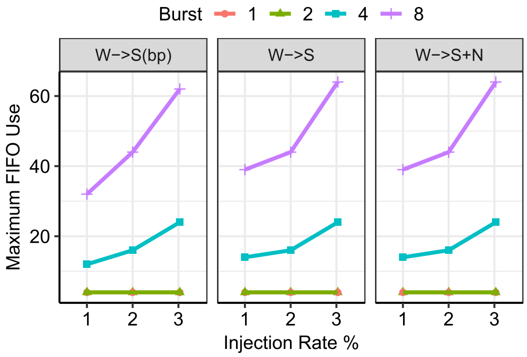


Fig. 23. Maximum FIFO usage trends from RTL simulations of NoCs with 5 × 5 system sizes for ALL-TO-ONE pattern.

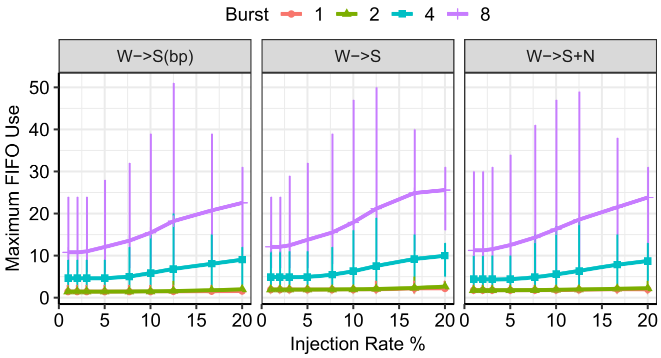


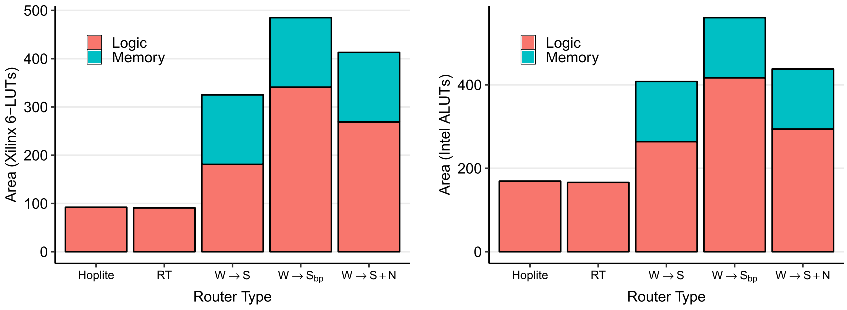
Fig. 24. Maximum FIFO usage trends from RTL simulations of NoCs with 5 × 5 system sizes for RANDOM pattern.

7.1.5 FIFO Sizing. Ultimately, the NoC with improved worst-case latencies is useful to us only if the buffer sizes are reasonable to realize on modern FPGAs. For a single LUT we can get 16–32 storage bits for our FIFOs, making it possible to build LUTs using these low-cost components. We cap our experiments at 128-deep FIFO sizes to keep NoC LUT cost at 4 LUTs/bit. For the ALL-TO-ONE traffic pattern shown in Figure 23, we will observe high FIFO usage in the column containing the destination client. As burst length increases, the FIFO usage also scales linearly with very low utilization with a burst length of 1–2. RANDOM traffic shown in Figure 24 exhibits slightly lower FIFO usage and demonstrates a spread of occupancy depending on connectivity pattern. While no experiment occupies more than 50 entries in the FIFO, on average, we only need ≈20–25 entries. We note an odd reduction in FIFO occupancy above the 10% injection rate. This is because an increasing subset of flowsets are not feasible with the 128-deep FIFO limit; i.e., FIFOs start going full.

7.2 FPGA Place-and-Route Results

Now we compare the FPGA implementation costs of HopliteBuf to other FPGA-overlay NoCs presented in this thesis. We quantify the LUT utilization of the various Hoplite routers in Figure 25. We present resource costs on Xilinx and Intel FPGAs.

For the HopliteBuf $W \rightarrow S + N$ design, we are essentially dividing the traffic into two buffers going up and down, and hence, the total distributed RAM capacity stays the same as it just split



(a) Xilinx Mapping

(b) Intel Mapping

Fig. 25. LUT utilization for logic and memory across various Hoplite routers on Xilinx and Intel FPGAs with Payload=64b and FIFO=64 deep.

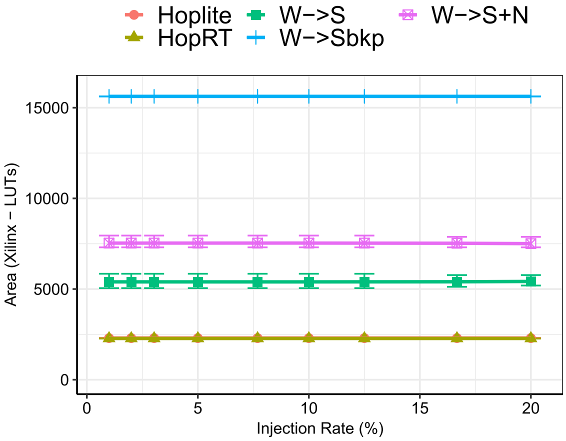


Fig. 26. Maximum area usage at different injection rates for RANDOM pattern with 5 × 5 system size. Each node in the system uses carefully picked FIFO sizes through static analysis in case of $W \rightarrow S$ and $W \rightarrow S + N$ HopliteBuf designs.

into two SRL32 or MLAB instantiations instead of a longer single distributed RAM block. Here, with limited opportunity for input sharing, the resulting design is larger in LUT cost, but as we have seen already, this allows efficient use of the NoC links.

As shown in the figure, the design size scales linearly with the product of Datawidth of the NoC × Depth of the FIFO on both vendor parts. With 64-deep FIFOs mapped to distributed RAMs, the storage fraction 🟦 increases design size by ≈2×.

The logic cost 🟥 varies with different router types. For the dual-FIFO HopliteBuf $W \rightarrow S + N$ design, the extra multiplexing needed for the upward route increases the cost of the switching logic. We observe the worst logic usage for $W \rightarrow S_{bp}$ backpressure design, where the classic flow-control logic implementation uses almost 2× more resources than other single-buffer designs. Both HopliteBuf designs achieve better performance than $W \rightarrow S_{bp}$ while maintaining low implementation costs.

In Figure 26, we show the area usage to implement a complete 5 × 5 system with different NoC designs. The experiment is done for 100 synthetic RANDOM flows where the prototype design, $W \rightarrow S_{bp}$, uses a fixed FIFO size of 128, whereas HopliteBuf ($W \rightarrow S$, $W \rightarrow S + N$) implements carefully

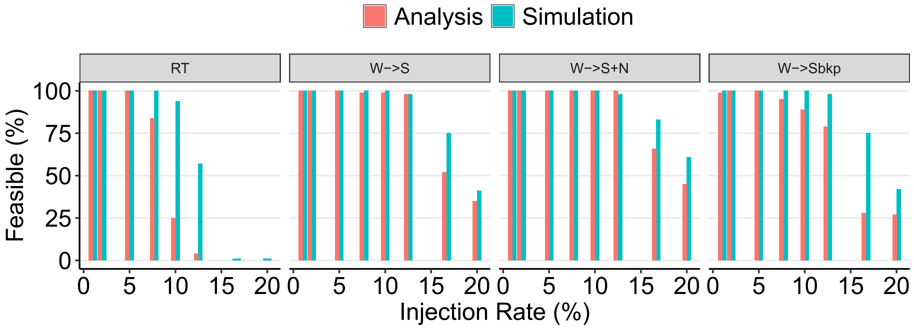


Fig. 27. Feasible flowsets predicted by static analysis. Analysis is more conservative than simulation for HopliteRT, but much tighter for HopliteBuf.

analyzed FIFO sizes by our analysis tool. As shown in the figure, HopliteBuf designs are ≈ 2–3× worse than Hoplite(RT) for all injection rates, whereas the $W \rightarrow S_{bp}$ design is ≈7 times worse than Hoplite(RT). This clearly shows the benefit of carefully analyzing the FIFO sizes for a particular application. Hence, we say that not only do the HopliteBuf designs outperform other NoCs in terms of latency and throughput, but also the overall hardware requirement is much better than the backpressure and deflection (fixed FIFO size) designs as well.

7.3 Analysis Results

We now examine the quality of our static analysis predictions and compare them to simulated data. The static analysis was only done for HopliteRT and the two variants of HopliteBuf: $W \rightarrow S$ and $W \rightarrow S + N$. Hence, we only compare the simulation and analysis results for these three router designs. We also consider analysis and simulation results for the $W \rightarrow Sbkp$ design with backpressure to illustrate the cost overhead and performance limitations of the conventional buffered design.

7.3.1 Feasible Flowsets. Our analysis tools take the communication pattern of a flowset and its injection rate ρ and burst b to determine if it can route successfully without making a FIFO ever go full. Analysis is more conservative, and you will note that Figure 27 is different from the simulation data in Figure 17. Our simulation results are for 1,024 packets per client, and there may be longer simulation conditions that ultimately become infeasible. Hence, we trust our analysis data as it is backed by the formal proofs explained in Section 6. Here, we see HopliteRT dropping dramatically above 8%, while HopliteBuf clones closely track simulation results. At 11% rates, we see analysis predict feasibility of only 2–3% for HopliteRT and ≈90% for HopliteBuf. Back in Figure 17, simulation results showed feasibility rates of 50% for HopliteRT and ≈90% for HopliteBuf. This suggests tighter analysis bounds for HopliteBuf resulting in better provable utilization of resources. When considering analysis results for feasibility, the $W \rightarrow Sbkp$ results are worse than $W \rightarrow S$ by 20–25%. Notably, the simulation results show that both designs perform equally well. This reflects pessimism in the analysis, which is a direct result of the complexities of interacting flows in the presence of backpressure.

7.3.2 Worst-Case Latency: Analysis versus Simulation. In Figure 28, we show the predicted worst-case latency count as a result of our static analysis versus actual observed latencies through simulation. As expected, the predicted bounds are worse with analysis due to pessimistic assumptions regarding interference of traffic flows. HopliteRT predictions are as much as 1.5× worse than the $W \rightarrow S + N$ predictions due to pessimism inherent in the HopliteRT routing algorithm. It is

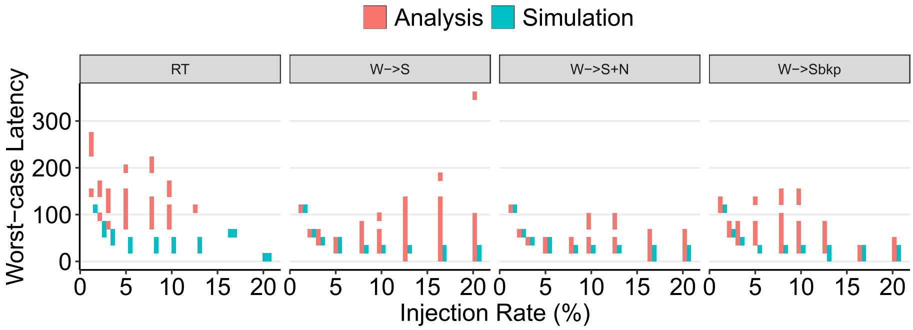


Fig. 28. Worst-case latency prediction versus simulation, RANDOM traffic, $b = 1$, 5×5 system size, 128-deep FIFOs.

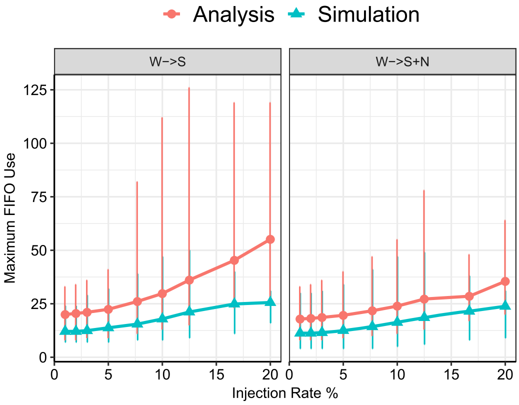


Fig. 29. Worst-case FIFO size prediction versus simulation for RANDOM traffic at 5×5 system size with 128-deep FIFOs and burstiness of 8.

interesting to see that a few flowsets mapped to HopliteRT at 16–20% injection rates actually simulate fine but are discarded by analysis as infeasible, yet again due to analytic pessimism. Furthermore, we see that the $W \rightarrow S + N$ predictions are significantly tighter than the $W \rightarrow S$ predictions. This is primarily due to the challenges associated with analyzing loopy flows in the vertical ring. When comparing the wider $W \rightarrow S$ spread to HopliteRT, it is important to note that a significant chunk of flowsets were infeasible when mapped to HopliteRT (see Figure 27). Thus, the larger latencies are due to $W \rightarrow S$ being able to feasibly route flowsets and doing so with high latencies than not being able to do so at all. The $W \rightarrow S + N$ analysis is significantly better than $W \rightarrow S$ and has a larger feasibility to compound the matter. For $W \rightarrow Sbkp$, the analysis latencies are larger than $W \rightarrow S$ at low injection rates ≤10%, but improved at higher injection rates >10%. At high injection rates, the $W \rightarrow S$ analysis tends to *blow up* close to link saturation limits. This is clear from Equation (16), where the leftover bandwidth along the link appears in the denominator. In contrast, $W \rightarrow Sbkp$ designs deliver better outcomes. When considering simulation results, again, both designs are pretty much equivalent.

7.3.3 FIFO Sizing: Analysis versus Simulation. In Figure 29, we compare the result of static analysis with simulated data for FIFO usage for a 5×5 NoC with RANDOM traffic and a worst-case burstiness of 8. For the $W \rightarrow S$ topology, we cap the maximum FIFO size to 128 to stay within a reasonable 4 LUTs/bit FIFO cost. In this case, the FIFOs go full for a few flowsets only above a

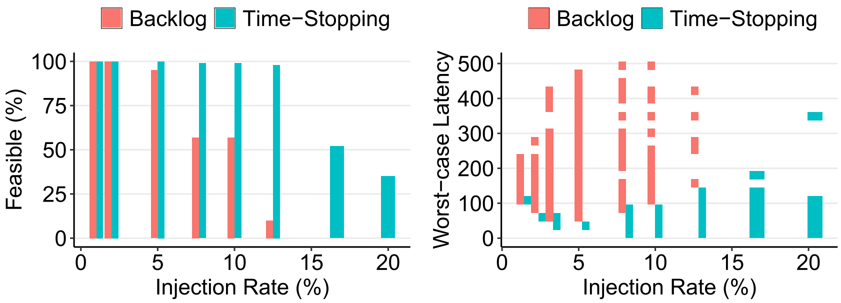


Fig. 30. Comparing Time-stopping and Backlog-based analysis methods.

healthy 15% injection rate. For the $W \rightarrow S + N$ topology, the FIFO sizes are capped at 64 (for a sum of 128) and only go full at a higher 20% injection rate. For both cases, we observe that simulated data shows lower occupancies than the prediction by as much as 2.5× (on average 1.5×). This is expected due the pessimism in the analysis, but the LUT cost impact is limited due to SRL32 packing quantization. For FPGA implementation, we can choose to size all FIFOs in the NoC to the largest size or customize each FIFO independently as per the static analysis. We observe that the largest size of 128 is rarely observed, and roughly 50% of our occupancies are below the 32 threshold. Thus, we can customize the right SRL depth to further save resources by as much as 2×.

7.3.4 Impact of Analysis Method ($W \rightarrow S$). Finally, in Figure 30, we compare the feasibility and worst-case latency trends for using the Time-stopping technique versus the Backlog-based bound for a 5 × 5 $W \rightarrow S$ NoC under Random traffic. As we saw previously in Theorem 3 from Section 6, there is a potential for supporting higher injection rates when using Backlog-based bounds at the expense of significantly worse latency values. However, the benefits of Backlog-based analysis seem to only be true for a one-off test example from Figure 13 in Section 6. In the case of randomly generated traces, we conclude there is no advantage to using a Backlog-based analysis technique for improving feasibility at higher rates. When considering feasibility, the Backlog-based method offers significantly lower feasibility and stops scaling beyond the 11% injection rate. This is primarily due to worst-case FIFO usage bounds exceeding the limit of 128. Furthermore, when considering worst-case latency, the spread of possible latency values exceeds the Time-stopping bounds by as much as 5×.

8 CONCLUSIONS

We present HopliteBuf, an FPGA-based NoC with lightweight buffering, and associated static analysis tools to better support NoC communication requirements of real-time FPGA applications. HopliteBuf introduces LUT-based stall-free FIFOs to the NoC router to absorb deflections and provide in-order routing of packets. We develop static analysis tools that can compute worst-case buffer occupancy bounds, along with latency bounds for communication patterns, with rate and burst information known up-front. In our experiments with 100 randomly generated flowsets, we show that HopliteBuf is able to deliver 40–50% feasibility at 20% injection rates, while the competing state-of-the-art HopliteRT NoC only supports 25% feasibility at 10% injection rates at 2× worse latency bounds. We also demonstrate 25–30% better feasibility outcomes compared to the conventional backpressure-based Hoplite variant at 30–40% lower LUT cost.

A APPENDIX

Table 9. DOR Routing Policy for $W \rightarrow S$ Router

Packet Paths					Muxsel		FIFO
$W \rightarrow E$	$W' \rightarrow S$	$N \rightarrow S$	$PE \rightarrow E$	$PE \rightarrow S$	Smx	Emx	**Read**
		x			00	-	
	x				01	-	1
x					–	0	
			x		–	1	
				x	10	-	
x		x			00	0	
		x	x		00	1	
x				x	10	0	
	x	x			00	-	0
	x		x		01	1	1
x	x				01	0	1

PE_i always has the least priority. S exit is shared with port PE_o exit. $W' \rightarrow S$ uses WFIFO read port. Extra combination of $W \rightarrow E$ and $W' \rightarrow S$ also supported.

Table 10. DOR Routing Policy for $W \rightarrow S + N$ Router

Packet Paths								Muxsel			FIFO	
$W \rightarrow E$	$W' \rightarrow S$	$W'' \rightarrow N$	$N \rightarrow S$	$Si \rightarrow N$	$PE \rightarrow E$	$PE \rightarrow S$	$PE \rightarrow N$	Smx	Emx	Nmx	**ReadS**	**ReadN**
			x					00	-	–		
	x							01	-	–	1	
				x				–	-	00		
		x						–	-	01		1
x								–	0	–		
							x	–	-	10		
						x		10	-	–		
					x			–	1	–		
x			x	x				00	0	00		
x		x	x					00	0	01		1
x			x				x	00	0	10		
			x	x	x			00	1	00		
		x	x		x			00	1	01		1
x	x			x				01	0	00	1	
x	x	x						01	0	01	1	1
x	x						x	01	0	10	1	
	x			x	x			01	1	00	1	
	x	x			x			01	1	01	1	1
x				x		x		10	0	00		
x		x				x		10	0	01		1

PE_i again has the least priority. S_o exit shared with port PE_o exit. $W' \rightarrow S$ uses WFIFO read port, $W'' \rightarrow N$ uses NFIFO read port.

Table 11. DOR Routing Policy for FIFO-W Backpressure Router

Packet Paths							Muxsel		FIFO	BP_{out}
$W \rightarrow E$	$W' \rightarrow S$	$N \rightarrow S$	$PE \rightarrow E$	$PE \rightarrow S$	BP_{in}	FULL	Smx	Emx	Read	
		x			1/0	1/0	00	-	-	0
	x				1/0	1/0	01	-	1	0
x					1	1/0	–	-	-	1
x					0	1/0	–	0	-	0
			x		1	1/0	–	-	-	0
			x		0	1/0	–	1	-	0
				x	1/0	1/0	10	-	-	0
x		x			1	1/0	00	-	-	1
x		x			0	1/0	00	0	-	0
		x	x		1	1/0	00	-	-	0
		x	x		0	1/0	00	1	-	0
x				x	1	1/0	10	-	-	1
x				x	0	1/0	10	0	-	0
	x	x			1/0	1	00	-	0	1
	x	x			1/0	0	00	-	0	0
	x		x		1/0	1/0	01	1	1	0
x	x				1	1/0	01	-	1	1
x	x				0	1/0	01	0	1	0

PE_i always has the least priority. S exit is shared with port PE_o exit. $W' \rightarrow S$ uses WFIFO read port. Extra combination of $W \rightarrow E$ and $W' \rightarrow S$ also supported. BP_{in} and BP_{out} denotes the backpressure input and output respectively and the routing is affected by the WFIFO FULL signal.

ACKNOWLEDGMENTS

This work has been supported in part by NSERC and CMC Microsystems. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors. We also thank Jan Gray for providing access to the Hoplite RTL baseline code.

REFERENCES

[1] Altera Corp. 2015. Arria 10 Core Fabric and General Purpose I/Os Handbook. Retrieved from https://www.altera.com/en_US/pdfs/literature/hb/arria-10/a10_handbook.pdf.

[2] Ahmed Amari and Ahlem Mifdaoui. 2017. Worst-case timing analysis of ring networks with cyclic dependencies using network calculus. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'17)*. IEEE.

[3] Ken Chapman. 2008. Saving costs with the SRL16E. *White Paper WP271 (v1. 0), Xilinx Inc* (2008).

[4] Jan Gray. 2016. GRVI-Phalanx: A massively parallel RISC-V FPGA accelerator accelerator. In *Proceedings of the 24th IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 17–20.

[5] Yutian Huan and A. DeHon. 2012. FPGA optimized packet-switched NoC using split and merge primitives. In *Field-Programmable Technology*. 47–52.

[6] S. Jeon, J. Cho, Y. Jung, S. Park, and T. Han. 2011. Automotive hardware development according to ISO 26262. In *13th International Conference on Advanced Communication Technology (ICACT'11)*. 588–592.

[7] N. Kapre and J. Gray. 2015. Hoplite: Building austere overlay NoCs for FPGAs. In *Field Programmable Logic and Applications*. 1–8. DOI: https://doi.org/10.1109/FPL.2015.7293956

[8] H. Kashif and H. Patel. 2014. Bounding buffer space requirements for real-time priority-aware networks. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC'14)*. 113–118.

[9] Hany Kashif and Hiren Patel. 2016. Buffer space allocation for real-time priority-aware networks. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'16)*. IEEE, 1–12.

[10] John Kim. 2009. Low-cost router microarchitecture for on-chip networks. In *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42'09)*, David H. Albonesi, Margaret Martonosi, David I. August, and José F. Martínez (Eds.). ACM, 255–266. DOI: https://doi.org/10.1145/1669112.1669145

[11] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag.

[12] Michael K. Papamichael and James C. Hoe. 2012. CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 37–46.

[13] Ian Swarbrick, Dinesh Gaitonde, Sagheer Ahmad, Brian Gaide, and Ygal Arbel. 2019. Network-on-chip programmable platform in VersalTM ACAP architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'19)*. ACM, New York, NY, 212–221. DOI: https://doi.org/10.1145/3289602.3293908

[14] Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. 2017. HopliteRT: An efficient FPGA NoC for real-time applications. In *2017 International Conference on Field Programmable Technology (ICFPT'17)*. IEEE, 64–71.

[15] Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. 2017. Worst Case Latency Analysis for Hoplite FPGA-based NoC. Retrieved from http://hdl.handle.net/10012/12600.

[16] Xilinx Inc. 2015. 7 Series FPGAs Configurable Logic Block User Guide. Retrieved from http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf.

Received May 2019; revised December 2019; accepted December 2019