HopliteBuf: FPGA NoCs with Provably Stall-Free FIFOs

Tushar Garg, Saud Wasly, Rodolfo Pellizzoni, Nachiket Kapre
{t3garg,swasly,rpellizz,nachiket}@uwaterloo.ca
University of Waterloo
Ontario, Canada

ABSTRACT

Deflection-routed NoCs like Hoplite and HopliteRT take advantage of FPGA-specific features to deliver low-cost, high-frequency, FPGA-friendly communication networks. However, they suffer from long packet deflection penalties, low sustained throughputs, and feature limitations such as out-of-order delivery of packets. In this paper, we introduce the HopliteBuf NoC, and an associated static analysis tool, that eliminates deflections entirely while simultaneously adding in-order delivery feature using (1) small, stall-free FIFOs with provable occupancy bounds, and (2) linearization of vertical rings of the torus Hoplite topology to improve provable link utilization. We implement these FIFOs using cheap LUT SRAMs (Xilinx SRL32s, and Intel MLABs) to absorb packet contention. We evaluate conditions for stall-free behavior using static analysis that compute upper bounds on FIFO occupancy based on the communication pattern. Our static analysis deliver bounds that are not only better (in latency) than HopliteRT but also tighter by 2–3×. 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 ≈2× for mutually feasible flowsets. At 20% injection rates, HopliteRT is only able to route 1–2% of the flowsets while HopliteBuf can deliver 40–50% sustainability.

ACM Reference Format:
Tushar Garg, Saud Wasly, Rodolfo Pellizzoni, Nachiket Kapre. 2019. HopliteBuf: FPGA NoCs with Provably Stall-Free FIFOs. In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19), February 24–26, 2019, Seaside, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3289602.3293917

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.

High-performance communication networks are vital for supporting connectivity requirements of modern FPGA designs. FPGA logic can be configured to implement packet-switched NoCs to allow IP blocks to interact with each other at the cost of stealing logic and routing resources away from the developer. In contrast, the system-level interface bandwidth requirements are driving FPGA vendors to embrace hardened NoC resources that bake in the network functionality without using any soft logic. Regardless of the choice of soft or hard NoC technology, 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 [4] requires performance isolation between communicating components on a chip and livelocks in NoCs violate this requirement. While timing analysis of statically-scheduled FPGA datapaths is simple, the analysis of communicating components using a shared dynamically-scheduled resource like a NoC is not so. NoC performance analysis is notoriously hard; it is often pessimistic and leads to over-provisioning of resources. In this paper, 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 and HopliteRT 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 1.2–1.3 ns 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.

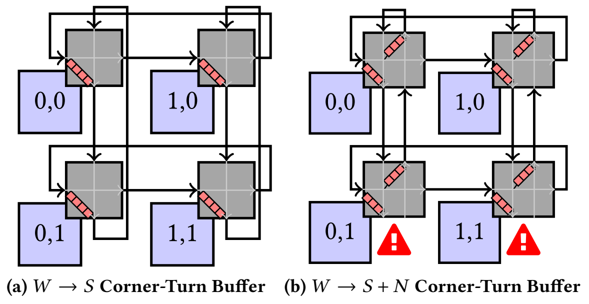

0,0
1,0
0,1
1,1
0,0
1,0
0,1
1,1
(a) $W \rightarrow S$ Corner-Turn Buffer
(b) $W \rightarrow S + N$ Corner-Turn Buffer

Figure 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FPGA '19, February 24–26, 2019, Seaside, CA, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6137-8/19/02…$15.00
https://doi.org/10.1145/3289602.3293917

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 analyze for static analysis of buffer bounds due to the complexity of packet interactions. Contemporary buffered FPGA NoCs like CMU CONNECT [12] and Penn Split-Merge [3] NoCs are very expensive and occupy 1000s of LUTs/router for 32-bit routers. Furthermore, the state-of-the-art analysis of NoC buffer bounds [6, 7] is pessimistic due to the complexity of modeling pipelining effects and mixing of various flows in the network.

In this paper, 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, (3) provides optional linearization of vertical NoC rings to enhance analysis that, (4) bounds buffer sizes to distributed-RAM friendly implementation sizes. The key contributions of our work is the microarchitecture of analysis-friendly HopliteBuf NoC routers, 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 1b 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 1a. 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 that 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, 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.

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.

Hoplite 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). This simple design requires a pair of 2:1 muxes as shown in Figure 2a. The DOR control logic is very

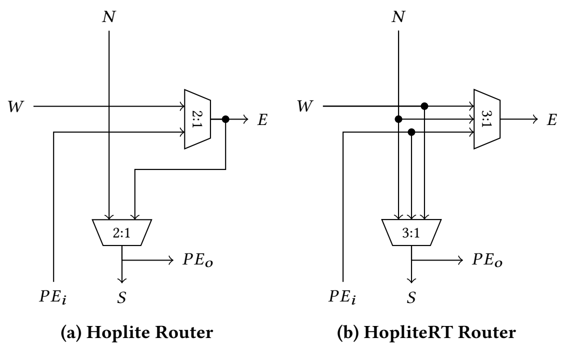


Figure 2: High-level design sketches of Hoplite (can livelock) and HopliteRT (no livelock) FPGA NoC Routers. HopliteRT adds a N→E turn to Hoplite to eliminate livelock. Both designs fit one bit of crossbar in one Xilinx 6-LUT.

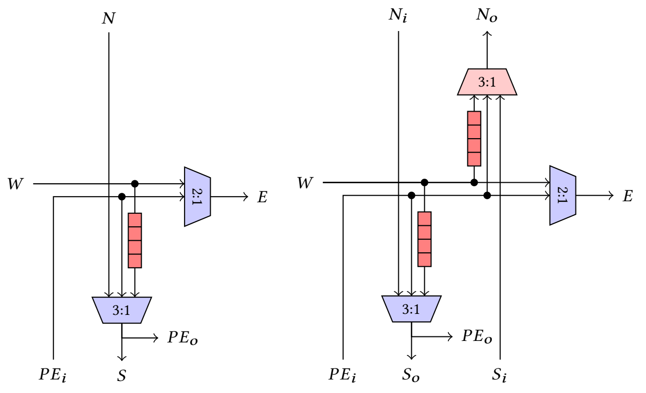
simple and consumes very few LUTs as it can be constructed directly on valid signals of the incoming packets alone. Packets exiting the NoC must do so over the S port to allow the mux and wires to be shared by both south-bound and exiting packets. A separate output valid signal helps determine the nature of the packet. The NoC client is provided the lowest priority and cannot inject a packet is 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 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 identical manner.

HopliteRT is a refinement over Hoplite that inverts the priorities and deflects N packet to the E (hence the new N→E turn in Figure 2b). Thus, HopliteRT requires two 3:1 muxes but still requires the same number of LUTs as Hoplite due to common multiplexer selection inputs to both muxes. HopliteRT overcomes the livelock limitation by forcing the N packet to deflect E and reappear 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 puts an upper bound on in-flight NoC latency to $\Delta X + \Delta Y \times m + 2$ for an $m \times m$ NoC. This indicates that, in the worst-case, the torus NoC could reduce down to a ring $O(m^2)$.

While HopliteRT costs the same as Hoplite and removes livelocks, it still suffers from an unusually high worst-case deflection bound while doing nothing to eliminate out-of-order delivery. Reducing a bandwidth-rich torus to a ring is neither an efficient nor scalable use of FPGA resources. Reordering of packets now becomes the responsibility of the NoC client and can add extra memory resources at the endpoints. In this paper, we propose develop a new NoC that not only preserve livelock-free behavior, but also provides improved latency bounds along with in-order packet delivery.

3 HOPLITE WITH STALL-FREE BUFFERS

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



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

Figure 3: 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.

3.1 The Idea

Earlier in Figure 1, we sketched two variants of the proposed HopliteBuf topologies with buffers for turning packets. In Figure 3, we show the switch microarchitectures of these variants. The basic multiplexing functionality implements turns to support 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 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 as well as 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:

- $W \rightarrow S$ **buffer**: In this scenario, packets turning from 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 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 the Section 3.2 and the restrictions of the routing combinations in Section 3.3. 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 negligible performance hit from this cost-saving transformation.
- $W \rightarrow S + N$ **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 1b. 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 well 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.

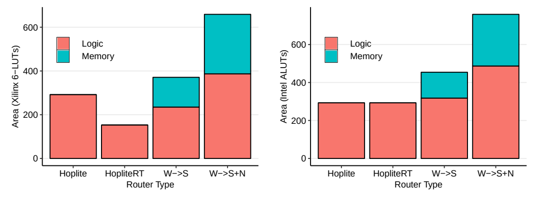
3.2 FPGA Implementation

A Xilinx 6-LUT is fracturable into two 5-LUTs with five common inputs across both LUTs. This allows you to implement one function of 5-inputs (any function) and one function of 6-inputs (if it overlaps with the 5-input function) in the same LUT. An Intel ALUT is organized differently and has 8-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 6-inputs as long as they share four inputs. Apart from logic, certain Xilinx LUTs can be programmed as 32-deep memories or shift registers. Intel offers a similar functionality with their MLAB (Memory Logic Array Block) resources. The stall-free buffer component of our new designs are implemented using these cheap resources.

Original Hoplite Mapping: When implementing the original Hoplite router shown in Figure 2a on a Xilinx FPGA, we can easily fit the two 2:1 muxes in two Xilinx 5-LUTs to allow a compact 1 6-LUT mapping per bit of switching. This is possible as the East 2:1 mux can use a 5-LUT (requiring 3 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 we can the 3:1 mux (South mux if implemented fully) needs 5 inputs while the 2:1 mux only needs 3. 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 inputs along with 3 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.

Mapping $W \rightarrow S + N$ FIFO Design This design requires the switching crossbar to consume six packet inputs: N_i, W, W' ($W \rightarrow S$ FIFO), W'' ($W \rightarrow N$ FIFO) and PE_i and 5 mux-select inputs. We choose to split the turning packets into separate FIFOs to prevent



(a) Xilinx Mapping **(b) Intel Mapping**

Figure 4: LUT utilization for logic and memory across various Hoplite routers on Xilinx and Intel FPGAs with Payload=128b and FIFO=32 deep.

mutual interference between traversing flows. The total distributed RAM capacity stays 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.

We quantify the LUT utilization of the various Hoplite routers in Figure 4. The design size scales linearly with the product of Datawidth of the NoC $\times$ Depth of the FIFO on both vendor parts. With 32-deep FIFOs mapped to distributed RAMs, the storage fraction 🟦 increases design size by 1.2–1.5×. For the dual-FIFO design, the extra multiplexing also increases cost of the switching logic 🟥.

3.3 Routing Policy

The original Hoplite and HopliteRT routers implemented bufferless deflection routing rooted in Dimension-Ordered Routing (DOR) 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 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 $N \longrightarrow E$ deflection that is not permitted under standard DOR implementation.

For HopliteBuf, we restore DOR routing policy but introduce extra decision logic for servicing FIFO packets. With buffering, W packets are forced to wait in the buffer thereby transferring priority to N port for $W \longrightarrow S$ variant, and to N_i and S_i packets for $W \longrightarrow S + N$ variant. All routers still accept PE packets with the least priority.

4 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 and in-flight traversal latencies. This is important to establish whether we can realize them in distributed FPGA RAMs (SRLs and MLABs). We first introduce 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 make this analysis susceptible to instability, but we are able to provide an analytic solution that employs a topology linearization alternative (Figure 1b) to eliminate cycles.

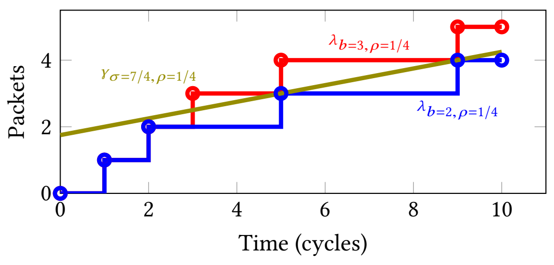


Figure 5: Example traffic curves for $\lambda_{b=3,\rho=1/4}$ and $\lambda_{b=2,\rho=1/4}$ along with an arrival curve for $\gamma_{\sigma=7/4,\rho=1/4}$.

4.1 Client Traffic Regulation

Injection regulation is a known technique from network calculus to establish well-defined behavior of network traffic at runtime for off-chip internet-scale systems. We adapt token bucket regulation [14] for use in an on-chip context 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. The regulator is programmed with a rate ρ and burst b that reflects the communication requirements of the application. At run-time, the regulator maintains a token counter. A packet can only be injected if the NoC is ready (no other packet is blocking the client) and there is at least one token in the counter; the token is consumed upon sending the packet. New tokens are generated and added to the counter at a rate ρ, provided that the counter has not saturated to its maximum value of b tokens.

4.2 Traffic Model

Definition 1. ***Traffic curve***: *To analyze the traffic characteristics, we introduce a traffic curve $\lambda_{b,\rho}(t)$ to denote the maximum number of packets sent on a NoC link in any interval of t cycles. By definition, a token bucket regulator with parameters b, ρ provides a traffic curve:*

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

Example: Traffic curves for two regulators with $b = 2, \rho = 1/4$ and $b = 3, \rho = 1/4$ are depicted in Figure 5 (the arrival curve γ will be introduced in Section 4.4). Consider the regulator with $b = 3$. 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. Afterwards, new packets are sent every $1/\rho$ cycles.

In this analysis, 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.

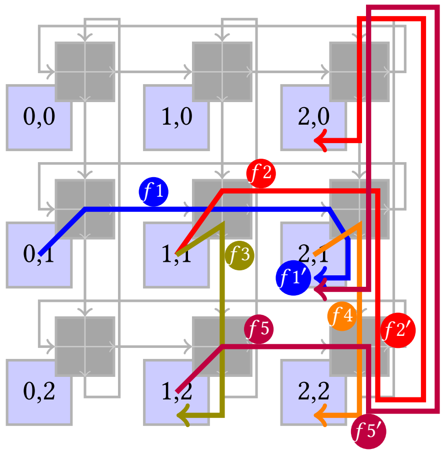


Figure 6: Example $W \to S$ NoC design with five flows $f_{1...5}$.

Table 1: Flow parameters for the example NoC. Γ^C are the conflicting flows used in Section 4.3; $f_{W\to S}$ and $f_{N\to S}$ are the $W \to S$ and $N \to S$ interfering flows used in Section 4.4. '-' denotes not applicable.

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	$f_2' + f_4$

Example: We present a running example of a NoC with five flows $f_{1...5}$ using the $W \to S$ buffer design in Figure 6. 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 1. We discuss how to apply the analysis to the $W \to S + N$ design in Section 4.5 as the flows have been linearized and have no loops. For the $W \to S$ design, the analysis is harder due to the loopback of the vertical ring. The instability created by loop-backs is a notoriously challenging problem in network calculus [9] and results in lower provable bounds on link utilization. We analyze the unique problem formulation presented by the HopliteBuf torus network and propose a technique for deriving these bounds and improving link utilization through linearization of the vertical ring.

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 turning flow f.
- 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).

4.3 Injection Latency

We first determine the set Γ^C of conflicting flows for f, that is, the set of flows that can block injection of a packet of f. It comprises:

- all other flows injected by the same source client, since a client can only inject a single packet per clock cycle;
- all flows originating from other clients that traverse the same mux used by f at its source router $(f.xs, f.ys)$.

Example: For flow f_2 in Figure 6, the conflicting set comprises flow f_3 (same client) and flow f_1 (E mux). For f_4, the set comprises flows f_1', f_2' and f_5' (S mux).

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 the burstiness parameters b of traffic curves for flows in Γ^C, and $\rho(\Gamma^C)$ as the sum of their rate parameters. Based on the token bucket regulator analysis provided in [14], we then obtain:

$$Injection(f) = \lceil 1/f.\rho \rceil - 1 + \left\lceil \frac{\sigma(\Gamma^C)}{1 - \rho(\Gamma^C)} \right\rceil. \quad (2)$$

Note that the condition $\rho(\Gamma^C) < 1$ implies that the cumulative rate of conflicting flows is less than 1 packet/cycle; this guarantees that packets of flow f are not permanently blocked at the source. Furthermore, if the client wishes to inject a sequence of $k > 1$ packets, it is possible to obtain an improved bound on the injection latency for the whole sequence as long as $k \leq f.b$. For simplicity we consider single-packet injection with $f.b = 1$.

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 $W \to S$ buffer in the next section.

4.4 Vertical Ring Analysis $W \to S$ Design

We now analyze the behaviour of flows turning on a vertical ring through a $W \to S$ buffer. We employ the theory of network calculus [9] for FIFO-arbitrated flows to derive deterministic bounds on 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 $N \to S$. To apply the theory, we need to introduce a new type of curves.

Definition 2. ***Leaky bucket arrival curve**: A flow f is said to the bounded by a leaky bucket arrival curve $\gamma_{\sigma,\rho}(t)$ if the number of packets transmitted by the flow in any time interval t is bounded by:*

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

In this case, $f.\sigma$ and $f.\rho$ are arrival curve parameters for the flow.

Luckily, we can convert between traffic curves of the form $\lambda_{b,\rho}(t)$ and arrival curves $\gamma_{\sigma,\rho}(t)$ according to the following lemma (a formal proof is provided in Lemma 1 in Appendix):

- to convert $\lambda_{b,\rho}(t)$ into $\gamma_{\sigma,\rho}(t)$, we set $\sigma = b - \rho$;
- to convert $\gamma_{\sigma,\rho}(t)$ into $\lambda_{b,\rho}(t)$, we set $b = \lceil \sigma + \rho + 1 \rceil$.

Example: Refer again to Figure 5. The traffic curve $\lambda_{b=2,\rho=1/4}(t)$ is upper bounded by $\gamma_{\sigma=7/4,\rho=1/4}(t)$. Similarly, arrival 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 1 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.

Finally, there are situations where we need to aggregate (combine) flows transmitted on the same link; for example, flows f_2' and f_4 entering router $(2,2)$ from N. 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.

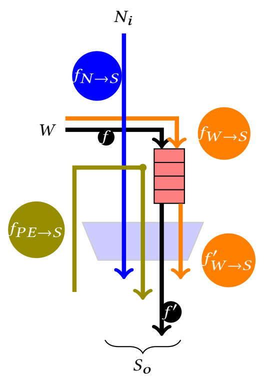


Figure 7: Flows through a $W \to S$ router.

Figure 7 illustrates the flows required for analysis at one NoC router. Here, f and f' represent a flow under analysis before and after leaving the $W \to S$ buffer; $f_{W\to S}$ represents the aggregate of all other interfering flows traversing the buffer; $f_{N\to S}$ represents the aggregate of all interfering flows traversing the router in the $N \to S$ direction; and $f_{PE\to S}$ represents the aggregate of all flows injected by the client at that router directly S. As discussed in Section 3.3, the S mux arbitration gives lowest priority to the client; hence, we do not have to consider flow $f_{PE\to S}$ when analyzing flow f, but it will interfere in the $N \to S$ direction on the next router. Regarding the other flows, $f_{N\to S}$ has higher priority than f, while $f_{W\to S}$ and f are FIFO scheduled as they traverse the same buffer.

Assuming that each flow is described by an arrival curve, we then obtain the following relations involving the curve parameters [1]:

$$f'.\rho = f.\rho; \tag{3}$$

$$f'.\sigma = f.\sigma + f.\rho \cdot \frac{f_{N\to S}.\sigma + f_{W\to S}.\sigma}{1 - f_{N\to S}.\rho}; \tag{4}$$

$$Backlog = f.\sigma + f_{W\to S}.\sigma + (f.\rho + f_{W\to S}.\rho) \cdot \frac{f_{N\to S}.\sigma}{1 - f_{N\to S}.\rho} \tag{5}$$

$$Delay(f) = \frac{f.\sigma}{1 - f_{N\to S}.\rho - f_{W\to S}.\rho} + \frac{f_{N\to S}.\sigma + f_{W\to S}.\sigma}{1 - f_{N\to S}.\rho} \tag{6}$$

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

Based on Equation 3, buffering does not increase the rate of flows. Furthermore, based on Lemma 1, for any flow f_i that has not been buffered, including flow f, we have $f_i.\sigma = f_i.b - f_i.\rho$. Hence, the only unknowns in Equation 4 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 [9]: we treat the values $f_i'.\sigma$ as variables, and write a system of linear equations by applying Equation 4 to each flow that enters a given vertical ring. If the values of $f_i'.\sigma$ obtained by solving the system of equations are valid (that is, bounded and positive), then $\gamma_{f_i'.\sigma, f_i'.\rho}(t)$ upper bounds flow f_i'. Otherwise, the network cannot be analyzed.

Example: Assume $f_1.\rho + f_2.\rho + f_5.\rho < 1$. For flow f_1, $f_{W->S}$ comprises flow f_2, while $f_{N->S}$ comprises flow f_5'. Since for any flow $f_i.\sigma = f_i'.\sigma$ and $f_i.\sigma = f_1.b - f.\rho$, 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 4 to flows f_2, f_5 under the added assumption $f_2.\rho + f_4.\rho + f_5.\rho < 1$ yields:

$$f_2'.\sigma = f_2.b - f_2.\rho + f_2.\rho \cdot (f_5'.\sigma + f_1.b - f_1.\rho)/(1 - f_5.\rho),$$
$$f_5'.\sigma = f_5.b - f_5.\rho + f_5.\rho \cdot (f_2'.\sigma + f_4.b - f_4.\rho)/(1 - f_2.\rho - f_4.\rho).$$

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 5, 6. Furthermore, by applying Lemma 1, 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 we add 1 to the buffer size to account for a packet being read from the buffer and transmitted in the current clock cycle.

Despite this, it is known [1, 9] that the circular dependencies introduced by a ring design can reduce the sustainable (provable) per-link utilization of the network by up to 50%. We now present the linearization alternative that overcomes this low peak utilization.

4.5 Linearized Analysis: $W \to S + N$ Design

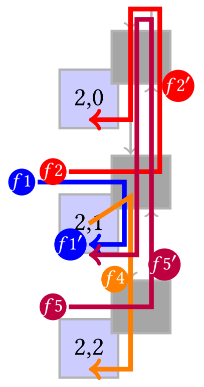


Figure 8: $W \to S + N$ design example: rightmost column.

The analysis for the $W \to S + N$ design proceeds in a similar manner, but is much simpler as no vertical loopback exists. The same injection latency computation is performed, albeit the set Γ_f^C can be different compared to the $W \to S$ design since a flow that was conflicting on the S mux could now turn N instead. Similarly, the same conditions in Equations 3-6 can be applied after decoupling each router in two parts: a south component containing the $W \to S$ buffer and S mux, and a north component containing the $W \to N$ 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 $f_{S\to N}$ and $f_{W\to N}$ in place of $f_{N\to S}$ and $f_{W\to S}$.

Example: Figure 8 shows the resulting decomposition for the rightmost column of the flow set depicted in Figure 6. Note that the

[1]In details, Proposition 1.3.4 in [9] is first used to determine a service curve for the aggregate of flows $f, f_{W\to S}$. Corollary 6.2.3 is then used to derive $f'.\rho, f'.\sigma$, as well as the service curve for f. Backlog and delay bounds follow from Theorems 1.4.1, 1.4.2.

Table 2: Conflicting and interfering flows for the $W \rightarrow S + N$ design. '-' denotes not applicable, as the flow is not buffered in that direction.

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

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_f^C and interfering flows $f_{N\rightarrow S}, f_{W\rightarrow S}, f_{S\rightarrow N}, f_{W\rightarrow N}$ are provided in Table 2. 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 4 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 N, and then from $(x, 0)$ back to $(x, m-1)$ for flows turning S. As long as no link is saturated, this guarantees that the analysis computes bounded delay and backlog.

4.6 Scaling regulation

For multiple flows starting from a client, we can design traffic regulators in two primary ways:

- We can implement a new token regulator for each outgoing flow by replicating counters per flow to deliver good analysis outcomes at increasing cost. For each outgoing flow per client, we need two counters to implement token regulation. This cost may be acceptable for a handful of outgoing flows.
- Alternatively, we can compute cumulative ρ and b parameters across all flows and program a single regulator. The rates and bursts are computed by summing the individual flow properties. This optimization will need a single regulator and single set of counters keeping costs low. However, this will result in pessimistic injection latency analysis due to interference.

5 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, as well as 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 use two synthetic workloads which are commonly used in the real-time systems community:

- We use `ALL-TO-ONE` pattern that gets all NoC clients to target a same NoC address: a shared resource like an external DRAM, PCIe, or Network port.
- We also use synthetic uniform `RANDOM` traffic pattern that is expressed 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 burstiness and rate which is increased until the links saturate.

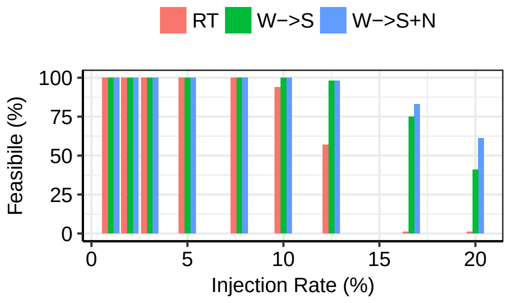


Figure 9: Feasible flowsets for `RANDOM` traffic with b=1 at 5×5 system size with 128-deep FIFOs in the NoC routers.

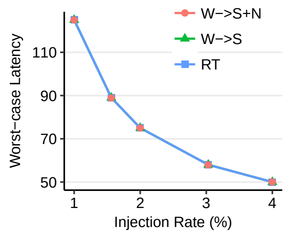
5.1 RTL Simulation Results

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

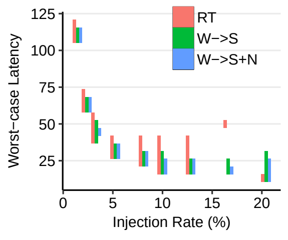
5.1.1 Flowset Feasibility. For our designs, we cap the maximum FIFO occupancy at 128 to enable low-cost realizations. As a result some combination of flowset communication pattern and injection rate ρ will likely be infeasible. If any FIFO ever goes full, we classify that configuration as not feasible. We want to know what fraction of our 100 randomly-generated flowsets were able to route without any of the NoC FIFOs every going full at a given rate.

In Figure 9, we plot the number of feasible flowsets for `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. 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 16% injection rates. The HopliteBuf NoCs 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 4.5, 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. Higher feasibility translates into more FPGA developer freedom in being able to support their communication requirements.

5.1.2 Worst-Case Latency Trends. We expect 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 10 we show this effect for two traffic patterns with burst b=1. 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



(a) ALL-to-ONE



(b) RANDOM

Figure 10: Worst-case latency trends for the different NoCs with 5×5 system sizes and b=1. HopliteBuf performs better for `RANDOM` pattern, and offers no improvements for `ALL-TO-ONE` traffic.

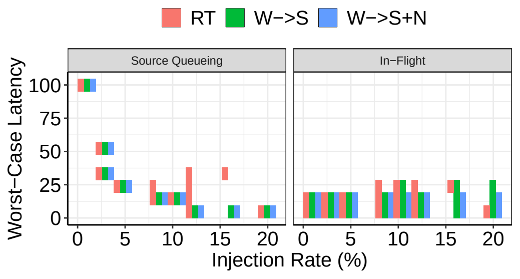


Figure 11: Breakdown of Source-Queueing and In-Flight NoC latencies for `RANDOM` workload with b=1 at 5×5 system size. Both metrics improved due to buffering.

is regulated and may miss the token cycle which 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 `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, $W \longrightarrow S$ designs suffer a buffer wait only at a single turn, it exhibits slightly worse performance that the two-FIFO $W \longrightarrow S + N$ 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 steals unnecessary bandwidth in the X-ring due to deflection.

5.1.3 Worst-Case Latency Breakdown. In Figure 11 we show a breakdown of worst-case latency into its source queueing (waiting time at PEs) and in-flight latency (actual routing time in the NoC). The improvements due to elimination of deflections does show up in better in-flight routing latencies, 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 HopliteRT routing scheme, the $N \longrightarrow E$ 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. HopliteBuf 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 presence of FIFOs.

5.1.4 Latency Distribution. In Figure 12, we show the histogram of worst-case packet latencies for the different NoCs for `RANDOM` traffic with burst b=1, and injection rate ρ=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. As expected, the $W \longrightarrow S$ design has a marginally wider distribution than the $W \longrightarrow S + N$ design as the packets have an extra choice during the turn.

5.1.5 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

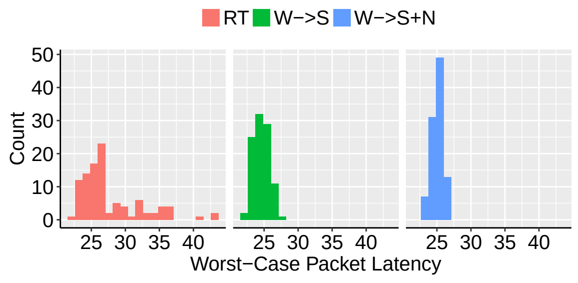


Figure 12: Distribution of worst-case packet latencies for `RANDOM` workload with b=1, ρ=7.5% at 5×5 system size. HopliteRT has a wider spread due to the unpredictable nature of the deflections. HopliteBuf has narrower spreads.

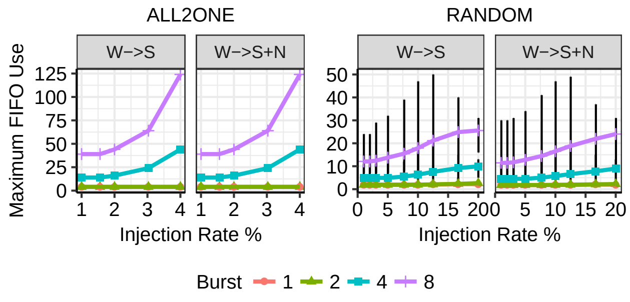


Figure 13: Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes. For `RANDOM` traffic, we observe a spread of maximum FIFO use as per pattern in the flowset, but lower than `ALL-TO-ONE` pattern.

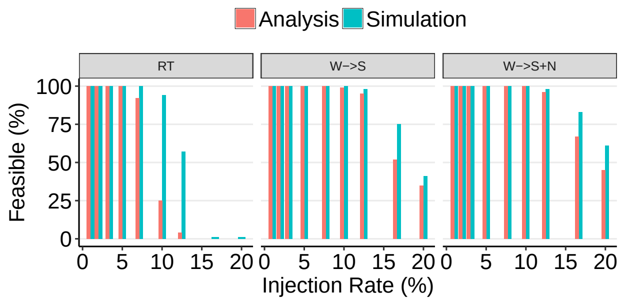


Figure 14: Feasible flowsets predicted by static analysis. Analysis is more conservative than simulation for HopliteRT, but much tighter for HopliteBuf.

sizes to keep NoC LUT cost at 4 LUTs/bit. For ALL-TO-ONE traffic pattern shown in Figure 13, 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 13 exhibits slightly lower FIFO usage and demonstrates a spread of occupancies 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 10% injection rate. This is because an increasing subset of flowsets are not feasible with 128-deep FIFO limit *i.e.* FIFOs start going full.

5.2 Analysis

We now examine the quality of our static analysis predictions and compare them to simulated data.

5.2.1 Feasible Flowsets. Our analysis tools take the communication pattern of a flowset, its injection rate ρ and burst b to determine if it is can route successfully without making a FIFO ever go full. Analysis is more conservative, and you will note that Figure 14 is different from the simulation data in Figure 9. Our simulation results are for 1K packets per client, and there may be longer simulation conditions that ultimately go infeasible. Hence, we trust our analysis data as it is backed by the formal proofs explained in Section 4. 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% of HopliteRT and ≈90% for HopliteBuf. Back in Figure 9, simulation results showed 50% of HopliteRT were feasible and ≈90% for HopliteBuf. This suggests tighter analysis bounds for HopliteBuf resulting in better provable utilization of resources.

5.2.2 Worst-Case Latency: Analysis vs. Simulation. In Figure 15, we show the predicted worst-case latency count as a result of our static analysis vs. 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 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

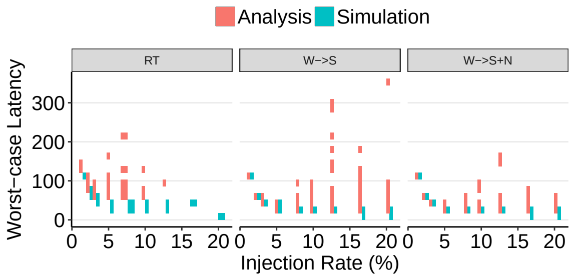


Figure 15: Worst-Case Latency Prediction-vs-Simulation, RANDOM traffic, b=1, 5×5 system size, 128-deep FIFOs.

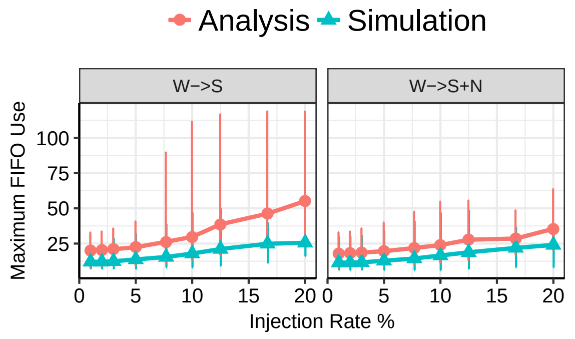


Figure 16: Worst-Case FIFO size Prediction-vs-Simulation for RANDOM traffic at 5×5 system size with 128-deep FIFOs and burstiness of 8.

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 14). 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.

5.2.3 FIFO Sizing: Analysis vs. Simulation. In Figure 16, 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 reasonable 4 LUTs/bit FIFO cost. In this case, the FIFOs go full for a few flowsets only above a 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×.

6 DISCUSSION AND RELATED WORK

Real-time NoC design has mostly focused in two directions:

- **Time-division multiplexed (TDM) NoCs** [2, 5, 11]: These NoCs require complete knowledge of the communication flows to build static TDM scheduling tables used for both packet injection and routing. However, this approach tends to be unsuitable for NoCs that need to support both real-time flows requiring worst case guarantees and best effort flows where average case delay matters.
- **Prioritized NoCs** [7, 13]: Here, packets are arbitrated based on the relative priority of the constituent flow instead of requiring complete knowledge. While this approach more effectively supports clients of different criticality, it requires expensive virtual channels: at each port, every priority flow needs a different buffer.

In this paper, we rely on a standard network calculus framework [9] to bound queuing delay and backlog so that we can guarantee the stall-free property of the buffers. The use of regulators to bound the maximum network latency is well-known in the context of network calculus. In particular, introducing a regulator at each router port can solve the dependencies issue in the $W \rightarrow S$ cyclic design, ensuring that the network remains stable up to full link utilization. Instead of expensive per-flow, per-router regulation popularly used in off-chip networks where cost of wiring is dominant, our work uses a cheaper per-flow regulator at injection.

PaterNoster NoC [10] is similar to Hoplite and uses a torus topology along with deflection routing. However, it is not optimized for FPGAs and uses many multiplexers and corner buffer. The use of corner buffer does help improve throughput but the NoC falls back to deflection routing when the buffers go full. In contrast, our NoC analysis guarantees stall-free behavior without resorting to deflections or affecting delivery order.

The **Kim NoC** router [8], uses separate X and Y rings like Hoplite, introduces an intermediate buffer for X→Y traversal, again like HopliteBuf, but uses backpressure flow control to manage full buffers. This topology uses multiplexers abundantly as it is targeted as ASICs, and would not match the LUT fabric of the FPGA as well as Hoplite or HopliteBuf. Furthermore, HopliteBuf does not require backpressure flow control either as FIFOs are not allowed to go full.

We anticipate our static analysis tools to help compute latency bounds on the newly-announced **Xilinx Versal NoC** with hardened FIFOs of known sizes. Unlike LUT-based HopliteBuf NoC used in this study, the Versal hard-NoC FIFOs do not need to be designed with LUT-FIFO capacity constraints.

7 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..

REFERENCES

[1] 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)*. IEEE.

[2] Kees Goossens and Andreas Hansson. 2010. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *47th DAC*. IEEE, 306–311.

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

[4] 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 (ICACT2011)*. 588–592.

[5] N. Kapre. 2016. Marathon: Statically-Scheduled Conflict-Free Routing on FPGA Overlay NoCs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 156–163. https://doi.org/10.1109/FCCM.2016.47

[6] 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)*. 113–118.

[7] 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)*. IEEE, 1–12.

[8] John Kim. 2009. Low-cost router microarchitecture for on-chip networks. In *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), December 12-16, 2009, New York, New York, USA*, David H. Albonesi, Margaret Martonosi, David I. August, and José F. Martínez (Eds.). ACM, 255–266. https://doi.org/10.1145/1669112.1669145

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

[10] Jörg Mische and Theo Ungerer. 2012. Low Power Flitwise Routing in an Unidirectional Torus with Minimal Buffering. In *Proceedings of the Fifth International Workshop on Network on Chip Architectures (NoCArc '12)*. ACM, New York, NY, USA, 63–68. https://doi.org/10.1145/2401716.2401730

[11] Jörg Mische and Theo Ungerer. 2014. Guaranteed Service Independent of the Task Placement in NoCs with Torus Topology. In *Proc. 22Nd RTNS. (RTNS '14)*. ACM, 151:160. https://doi.org/10.1145/2659787.2659804

[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] Zheng Shi and Alan Burns. 2008. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Second ACM/IEEE NOCS (nocs 2008) (NOCS '08)*. IEEE, 161–170. https://doi.org/10.1109/NOCS.2008.4492735

[14] Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. 2017. HopliteRT: An efficient FPGA NoC for real-time applications. In *F. Program. Technol. (ICFPT), 2017 Int. Conf.* IEEE, 64–71.

A APPENDIX

Lemma 1. *(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:

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

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 a NoC link cannot transmit more than one packet every clock cycle, the flow is bounded by:

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

□