# Worst Case Latency Analysis of Hoplite FPGA-based NoC

[1,2]Saud Wasly, [1]Rodolfo Pellizzoni, [1]Nachiket Kapre
[1]University of Waterloo, Canada, {swasly, rpellizz, nachiket}@uwaterloo.ca
[2]King Abdulaziz University, Saudi Arabia, swasly@kau.edu.sa

***Abstract*—**

**Overlay NoCs, such as Hoplite, are cheap to implement on an FPGA but provide no bounds on worst-case routing latency of packets traversing the NoC due to deflection routing. In this paper, we show how to adapt Hoplite to enable calculation of precise upper bounds on routing latency by modifying the routing function to prioritize deflections, and by regulating the injection of packets to meet certain throughput and burstiness constraints. We provide an analytical model for computing end-to-end latency in the form of (1) in-flight time in the network $T^f$, and (2) waiting time at the source node $T^s$. To bound in-flight time in an $m \times m$ NoC, we modify the routing function and switching crossbar richness in the Hoplite router to deliver $T^f = \Delta X + \Delta Y + (\Delta Y \times m) + 2$ where $\Delta X$ and $\Delta Y$ are differences of the source and destination address co-ordinates of the packet. To bound the waiting time at the source, we add a Token Bucket regulator with rate $\rho_i$ and burstiness $\sigma_i$ for each flow $f_i$ of node $(x, y)$ to deliver $(\lceil \frac{1}{\rho_i} \rceil - 1) + T^s$ : $T^s = \lceil \frac{\sigma(\Gamma_f^C)}{1-\rho(\Gamma_f^C)} \rceil$ which depends on the regulator period $1/\rho_i$, burstiness $\sigma$ and the rate $\rho$ of all interfering flows $\Gamma_f^C$. A 64b implementation of our HopliteRT router requires ≈4% fewer LUTs, and similar number of FFs compared to the original Hoplite router. We also need two small counters at each client port for regulating injection. We evaluate our model and RTL implementation across synthetic traffic patterns and observe behavior that conforms with the analytical bounds.**

`https://git.uwaterloo.ca/watcag-public/hoplitert-bounds`

## I. Introduction

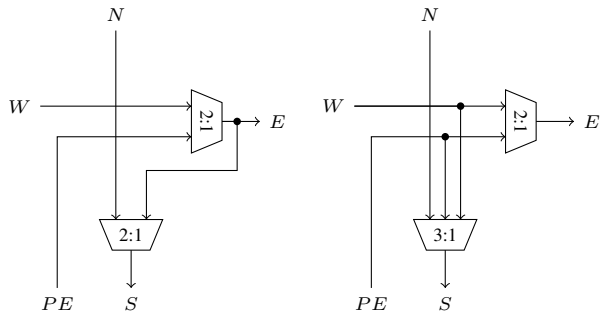
Overlay NoCs (network-on-chip) such as Hoplite [1] provide a low-cost, high-throughput implementation of communication networks on FPGAs. NoCs allow designers to compose large-scale multi-processor, or multi-IP digital systems while providing a standard communication interface for interaction. This trend is true in the embedded, real-time computing domain with multi-core chips supported by message-passing NoCs [2], [3]. Modern real-time applications [4], [5] require many communicating processing elements to cooperate on executing the task at hand. In contrast, with shared memory systems that rely on cache coherency, explicit message passing on a suitably-designed NoC allows real-time applications to have (1) deterministic bounds on memory access, and (2) energy-efficient transport of data within the chip. FPGAs should be a particularly attractive target for the real-time computing market due to the small shipment volumes of real-time products, and the ability to deliver precise timing guarantees that are desirable for certification and correct, safe operation of such systems.

Until recently, FPGA-based NoCs were inefficient and bloated due to the implementation cost of switching multiplexers and FIFO queues. Hoplite [1] uses deflection routing to (1) reduce the LUT mapping cost of the switching crossbar, and (2) remove queues from the router that are particularly expensive on FPGAs. When compared to other FPGA NoCs such as CMU Connect [6], and Penn Split-Merge [7] routers, Hoplite is lean, fast, and scalable to 1000s of routers on the same FPGA chip. Deflection routing resolves conflicts in the network by intentionally mis-routing one of the contending packets. In baseline Hoplite, these deflections can go on forever resulting in livelocks. Even if this is unlikely, such behavior makes it unsuitable to use this NoC in a real-time environment where bounds on routing time must be computable to meet strict timing deadlines imposed by the application. An age-based routing scheme for resolving livelocks in deflection-routed networks does exist, but this requires extra wiring cost for transporting age bits and still does not deliver strict upper bounds that we are able to show in this work.

In this work, we answer the following question: **Can we modify Hoplite for real-time applications to deliver strong, deterministic upper bounds on worst-case routing latency including waiting time at the client?** We introduce Hoplite-RT (Hoplite with Real-Time extensions) that requires no extra LUT resources in the router and only two cheap counters in the client/processing element. These modifications implement the new routing logic and packet regulation policy in the system. The router modification has a zero cost overhead over baseline Hoplite due to a simple re-encoding of multiplexer select signals that drive the switching crossbars. With cheap counters at the client injection port, we can enforce a token bucket injection policy that controls the burstiness and throughput of the various packet flows in the system.

The key contributions of our work include:

- Design and RTL implementation of Hoplite-RT routers to (a) evaluate resource costs on a Xilinx Virtex-7 FPGA, and (b) confirm correctness of the routing policy in cycle-accurate simulations.
- Computation of analytical bounds for (a) in-flight time in the NoC, and (b) waiting time at the client/PE ports.
- Design of an analysis tool for use in real-time system design flow that accepts a list of traffic flows to determine feasibility, and to calculate bounds using our analytical expressions. Validation of these bounds on cycle-accurate




(a) Shared East exit (b) Without sharing

Fig. 1: Implementation choices for the Hoplite FPGA NoC Router. A LUT-economical version (left) is able to exploit fracturable Xilinx 6-LUTs to fit both 2:1 muxes into a single 6-LUT. The larger, higher-bandwidth version (right) needs 2 6-LUTs instead as the number of common inputs is lower than required to allow fracturing.

simulations of various test cases.

## II. Motivation

In this section, we introduce the Hoplite router switching architecture and identify why the worst-case deflection and source queueing latencies can be unbounded.

Hoplite routes single-flit packets over a switched communication network using Dimension Ordered Routing (DOR). DOR policy makes packets traverse in the X-ring (horizontal) first followed by the Y-ring (vertical). Hoplite uses bufferless deflection routing and a unidirectional torus topology to save on FPGA implementation cost. While DOR is not strictly required for deflection-routed switches, Hoplite includes this feature to reduce switching cost by eliminating certain turns in the router. The internal microarchitecture of the router is shown in Figure 1 with three inputs $N$ (North), $W$ (West) and $PE$ (processing element or client, used interchangeably in the text) and two outputs $S$ (South + processor exit) and $E$ (East). When packets contend for the same exit port, one of them is intentionally mis-routed along an undesirable direction to avoid the need for buffering. The fractured implementation (Figure 1a) serializes the multiplexing decisions to enable a compact single Xilinx 6-LUT realization of the switching crossbar per bit. However, this sacrifices the routing freedom to achieve this low cost. A larger design (Figure 1b) that needs 2 6-LUTs per bit (2× more cost) permits a greater bandwidth through the switches. The larger cost is due to the inability to share enough inputs that would have allowed fracturing the Xilinx 6-LUT into dual 5-LUTs. This larger design will become important later in Section III when considering the microarchitecture of HopliteRT.

When considering the routing function capabilities of the Hoplite router, we make the following observations:

- The $PE$ port has lower priority than the network ports $N$ and $W$ resulting in waiting time for packet injections. A client can get dominated by traffic from other clients potentially blocking it forever. This is the source of unbounded waiting time at the client injection and called **source queueing latency**.
- The $N$ packet only travels $S$ and no path to $E$ is permitted under DOR routing rules. Thus $N$ packets can never be misrouted (or deflected) and are guaranteed unimpeded delivery to their destination.
- As a consequence, conflicts and deflections may only happen on a $W \rightarrow S$ packet which is attempting to turn. A deflected packet must route around the entire ring in the network before attempting a turn again.
- Due to the static priority for $N \rightarrow S$ packets, an unlucky $W \rightarrow S$ packet may deflect endlessly in the ring and livelock and result in unbounded NoC **in-flight routing time**. This scenario is shown in Figure 2.
- Deflections also steal bandwidth away from PEs in the ring, and add more waiting time penalty for packets wishing to use the $PE$ exit.

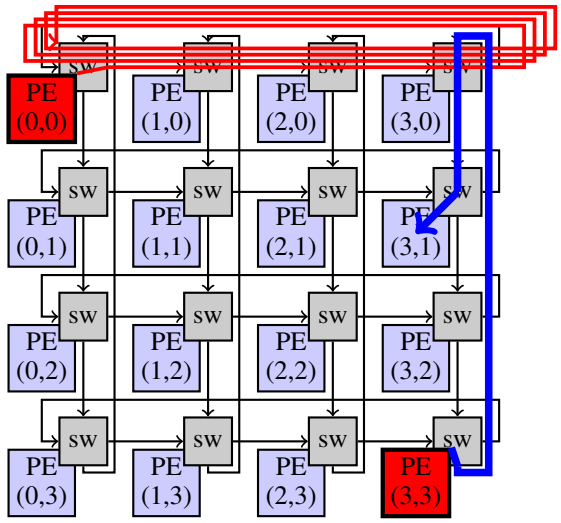



Fig. 2: Endless deflection scenario where red packets from (0,0) → (3,3) are perpetually deflected by blue packets from (3,3) → (3,1). The red spaghetti is the flight path of one packet that gets trapped in the top-most ring of the NoC and never gets a chance to exit due to the bossy blue packets.

To summarize, the communication latency between two PEs at $(X_1, Y_1)$ and $(X_2, Y_2)$ on the zero-load network is $T^s + (\Delta X + 1) + (\Delta Y + 1)$ due to DOR policy. Here, $T^s$ is the waiting time at PE level, $(\Delta X + 1) + (\Delta Y + 1)$ are the number of steps in the X-ring and the Y-ring respectively. **The worst-case in-flight routing and source queueing latencies in Hoplite are both** $\infty$. This is problematic for real-time applications that need guaranteed bounds on application execution. If such an application wishes to use the baseline Hoplite NoC for interacting with other components, one of the packets may get victimized and deflect endlessly or a client port may get blocked forever. As a result, the real-time application will miss its deadline and violate the system design requirements. For safety critical applications (hard real-time applications) the deadlines are hard constraints that cannot be violated. For NoCs in general, it is possible to provide statistical guarantees

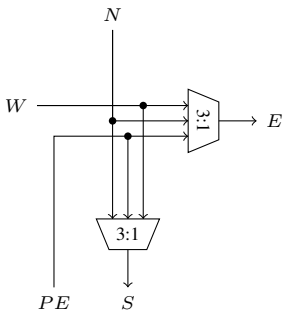



Fig. 3: Proposed router configuration for bounded in-flight latency. Despite splitting the logic into 2× 5-LUTs (3:1 muxes), the same multiplexer select signals (with different interpretation) drive both multiplexers. This allows a compact 6-LUT implementation per bit.

on packet delivery times but these are still not strong enough for hard real-time problems.

The next two sections proposes minimal modifications to the Hoplite router and Client (PE) to provide exact upper bounds of the worst-case latency of packet routing, and the worst-case waiting time at the source. These bounds can then be used by the real-time application developer to satisfy the hard deadline constraints imposed by the system.

## III. Managing In-Flight Deflections

In this section, we describe the modifications to the Hoplite router to support bounded deflections in the network. We explain the resulting routing table modifications that are required and explain the operation of the NoC with an example.

In Figure 3, we show the proposed microarchitecture of HopliteRT. The key insight here is the need to strategically introduce deflection freedom by making the switching crossbar more capable without sacrificing LUTs in return. The fractured LUT dual 2:1 LUT implementation in Figure 1a is too restrictive to permit any adaptations to the routing policy, and hence we consider a more capable dual 3:1 mux microarchitecture instead that may need the more expensive two 6-LUT implementations. This 3-input, 2-output crossbar arrangement permits any input to be routed to either output as desired by the routing policy. Thus, unlike the original Hoplite designs in Figure 1, we are able to use a $N \rightarrow E$ turn now to support our goals.

With this rich switching crossbar, we must now choose our routing policy. An age-based routing prioritization (Oldest-First [8]) can be implemented that prefers older packets over newer packets when in conflict. This is a good use of the $N \rightarrow E$ path which is exercised if $W \rightarrow S$ packet is older in age than the $N \rightarrow S$ packet. In this conflict situation the $N$ packet can be deflected $E$. The original $\infty$ bound will be reduced to a different bound that depends on the network size $m \times m$ and congestion or load in the system. However, this policy is unfair as it is biased towards traffic travelling from distant nodes as traffic from nearby closer nodes is always victimized. A variation of the policy that increments age only on deflections may be slightly fairer but the resulting bound is still dependent on network congestion. Furthermore, we need to transmit extra bits in each packet to record age of the packet which is wasteful of precious interconnect resources.

It turns out that we can limit the number of deflections without carrying any extra information in the packet. The key idea is to invert the priorities of the router to always prefer $W$ traffic over $N$ traffic. This is the exact opposite of the original Hoplite DOR policy that always prioritizes $N$ traffic (due to the absence of the $N \rightarrow E$ link). This modified policy allows traffic on the X-ring to be conflict free, even when making the turn to the Y-ring. On the other hand, it is the Y-ring traffic (North) that can suffer deflections. Once the Y-ring traffic has been deflected onto the X-ring, it will have a higher priority over any other Y-ring traffic it will encounter next. Thus, unlike the original design where packets deflects multiple times on the same row without making progress, now the deflected packet is guaranteed to make progress toward its destination. A packet might deflect only once on each row (X-ring).

The routing table for this new HopliteRT router is shown in Table I. The added benefit of this routing policy is the ability to use the exact same select bits for both 3:1 multiplexers. So not only do we bound deflections in the NoC, but we also ensure we can retain fracturability of the 6-LUT by supplying identical five inputs to the 3:1 mux. Each mux interprets the same two select bits differently to implement the proper routing decision. In this arrangement, the PE→E with W→S turns happening in the same cycle are not supported even though the mux bandwidth is rich enough to support this condition. This is because we want to avoid creating a third mux select signal that would prevent the fractured 6-LUT mapping. If the developer can afford to double the cost of their NoC, then this extra function can be supported without affecting the in-flight NoC worst-case routing time bounds computed in this paper. It is important to note that the bandwidth capability of the HopliteRT switch is in-between Figure 1a and Figure 1b. When compared to Figure 1a, HopliteRT supports an extra routing condition PE→S + W→E which is otherwise blocked due to cascading of the muxes. However, unlike Figure 1b, HopliteRT does not support PE→E + W→S condition. Thus, HopliteRT is strictly the same size as a less-capable switch in Figure 1a while offering extra bandwidth and latency guarantees.

TABLE I: Routing Function Table to support Real-Time extensions to 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.

| Mux select | | Routes | Explanation |
|---|---|---|---|
| sel1 | sel0 | | |
| 0 | 0 | W→E + N→S | Non-interfering |
| 0 | 1 | W→S + N→E | Conflict over S |
| 1 | 0 | PE→E + N→S | No W packet |
| 1 | 1 | PE→S + W→E | No N packet |

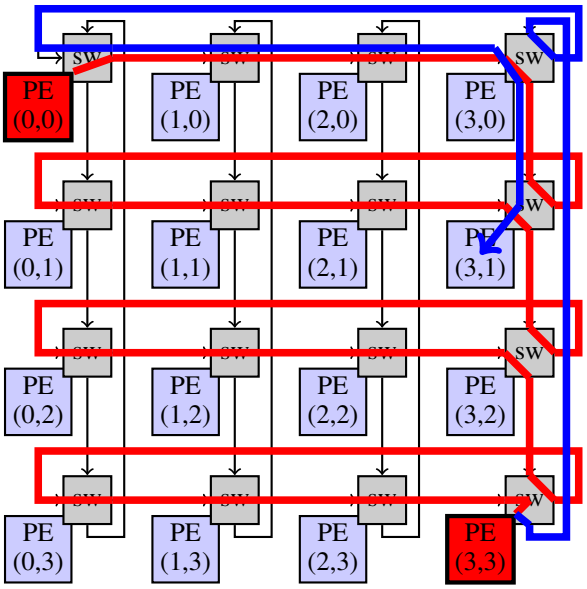



Fig. 4: Worst-Case path on Hoplite-RT for packet traversing from top-left PE (0,0) to bottom-right PE (3,3). The red packets will deflect N→E in each ring due to a conflicting flow (not shown). The blue packets previously had priority are now deflected in the top-most ring before delivery.

In Figure 4, we show the path taken by the packet from (0,0)→(3,3) as the example earlier in Figure 2. In this scenario, we assume there are $W \rightarrow S$ flows in X-ring 0, 1, and 2 that will interfere with the red packets in each ring. These flows will have priority over the $N \rightarrow S$ red packet and will deflect those red packets in each ring. The blue packet from (3,3)→(3,1) had the right of way in the original Hoplite NoC as shown in Figure 2. In HopliteRT, it will be deflected once in the topmost ring, and then descend downwards to exit at its destination.

**Analytic In-Flight Latency Bound**: Under the proposed HopliteRT policy, the in-flight latency between nodes $(X_1, Y_1)$ and $(X_2, Y_2)$ on an $m \times m$ NoC is as follows:

$$\textit{zero load: } T^f = \Delta X + \Delta Y + 2 \quad (1)$$

$$\textit{worst case: } T^f = \Delta X + \Delta Y + (\Delta Y \times m) + 2 \quad (2)$$

Here, $\Delta X = (X_2 - X_1 + m)\%m$ and $\Delta Y = (Y_2 - Y_1 + m)\%m$ are based on traversal distances of the packet on the torus irrespective of relative order of the two nodes along the directional topology. The zero load latency on the NoC is the same as original Hoplite.

Finally, in Table II, we show the effect of compiling Hoplite and HopliteRT to the Xilinx Virtex-7 485T FPGA and observe a minor 4% reduction in LUT costs as (1) the design is able to exploit the fracturable dual 5-LUTs per bit of the switching crossbar, and (2) the DOR logic is marginally simpler.

TABLE II: FPGA costs for 64b router (4×4 NoC) with Vivado 2016.4 (Default settings) + Virtex-7 485T FPGA

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

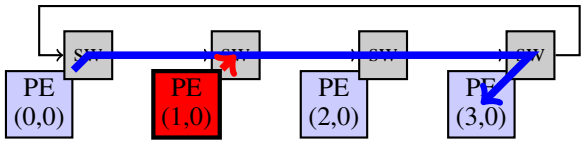



Fig. 5: Unlucky client at (1,0) is swamped by client at (0,0) that has flooded the NoC with packets at full link bandwidth (one packet per cycle). Red packets from (0,0)→(x,y) are perpetually blocking the client exit at (1,0). This results in a waiting time of $\infty$ for packets at (1,0)

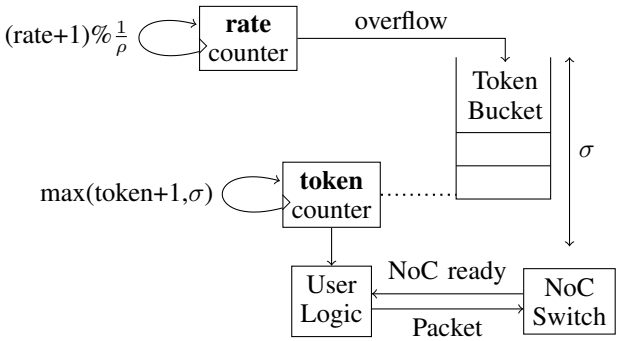



Fig. 6: Conceptual view of the Token Bucket regulator at the injection port of each NoC client. FPGA implementation cost is two cascaded counters (no actual memory is needed to store any tokens). The client can inject a packet into the NoC only when the NoC is ready and there is at least one token available in the Token Bucket.

## IV. Regulating Traffic Injection

While the HopliteRT modification to the original Hoplite router was able to bound in-flight NoC latency, source queueing delays can still be unbounded. Source queueing delays are attributed to the least priority assigned to client injection port by the Hoplite router. This is unavoidable in a bufferless setup where we do not have any place to store a packet that may get displaced if the client port was prioritized. Furthermore, when using deflection routing, there is no mechanism to distribute congestion information to upstream clients to throttle their injection. Hence, a client may wait arbitrarily long if it is swamped by another upstream client that has decided to flood the NoC with packets; a simple scenario is depicted in Figure 5. Here, we have two flows: the blue flow from (0,0)→(3,0), and a red flow from (1,0)→($x$,0). If the rate of the blue flow is 1 (one packet per cycle), the red flow will never get an opportunity to get onto the NoC. This is the cause of the $\infty$ waiting time at a client port. In this section, we discuss a discipline for regulating traffic injection that ensures bounded source queueing times in the bufferless, deflection-routed NoC such as Hoplite.

### A. Token Bucket Regulator

In Figure 6, we show a high-level conceptual view of a Token Bucket regulator [9] inserted on the client→NoC interface. The regulator shapes the traffic entering the network and bounds the maximum amount of time the client (user logic) will have to wait to send a packet over the network. The

regulator is defined by parameters $\rho$ (rate of packet injection $<$1 packet/cycle) and $\sigma$ (maximum burst of consecutive packets $\geq$1). While conventional token bucket implementations in large-scale network switches (*i.e.* internet) need to queue or drop packets, we assume that we are able to directly stall or throttle the source (client).

The regulator is implemented using two counters that are cheap to realize on an FPGA. A free-running **rate counter** is programmed to overflow after the regulation period $\frac{1}{\rho}$. On each overflow a token is added to an abstract bucket. The bucket will fill until it reaches its capacity $\sigma$ where is saturates. This is tracked by the second counter **token counter**. Whenever a client wants to inject traffic into the NoC, the client is stalled until the bucket is not empty and the NoC is ready (that is, no other NoC packet is blocking the client). The client then withdraws one token from the bucket and sends the packet.

As an example, if a client has a rate $\rho = \frac{1}{10}$, it can inject one packet every 10 cycles. The burstiness parameter $\sigma$ determines the maximum number of packets that the client can send consecutively, assuming that the NoC is ready. If the client has a burst length $\sigma = 5$, and it did not send any packets for 50 cycles, it will accumulate five tokens. As a result, it can send 5 packets in a burst before the token bucket is emptied.

It is worth mentioning that, in the worst case, if the traffic in the network is not regulated (unknown traffic) the entire network will be considered as one ring. Therefore, all the clients will have to share the bandwidth utilization $\sum \rho = 1$. In this case the aggregated injection rates for each client needs to be $1/m^2$ to guarantee injection for everybody; otherwise, a client will be starved by the upstream traffic indefinitely, and the bound on the waiting time will be unknown.

Based on the described regulation mechanism, in the next Section IV-B we compute the maximum time that the client requires to send a sequence of $k$ packets, with $k \leq \sigma$. We first bound the amount of network traffic that interferes with the injection of packets on either the East or South port. Then, we express a tight bound on the maximum amount of interference to ensure that the client is not starved. Assuming that the bound is met, we finally determine the maximum network delay for the injection of the first packet, and of any successive packet in the sequence.

### B. Analysis

We consider an $m \times m$ matrix of clients $(x, y)$. We generalize the earlier discussion in Section IV-A by assuming that $(x, y)$ can send packets to multiple other clients in the network, using a different token bucket regulator for each such client. Hence, for any client $(x, y)$, we can define a set of flows:

$$F_{x,y} = \{(x_1, y_1, \rho_1, \sigma_1), (x_2, y_2, \rho_2, \sigma_2), ..., (x_n, y_n, \rho_n, \sigma_n)\}, \tag{3}$$

where for flow $f_i = (x_i, y_i, \rho_i, \sigma_i)$, the destination client is $(x_i, y_i)$ and the token bucket parameters are $\rho_i, \sigma_i$. For a flow $f \in F_{x,y}$, we use the notation $f.x, f.y, f.\rho, f.\sigma$ to denote the horizontal, vertical coordinates and regulator parameters of the flow, respectively. We further use the notation $\lambda(t)$ to denote a **traffic curve**, that is the maximum number of packets transmitted on a port in any window of time of length $t$ cycles. By definition of the token bucket regulator, the maximum number of packets injected by $(x, y)$ with destination $(x_i, y_i)$ is then:

$$\lambda_{x,y}^{\sigma_i,\ \rho_i}(t) = \min\left(t,\ \sigma_i + \lfloor \rho_i \cdot (t-1) \rfloor\right). \tag{4}$$

To determine the total traffic that could pass through a client and possibly affect its injection rate, we need to know the aggregated traffic of the different sources that can reach to the client. Consider two traffic curves $\lambda^{\sigma_1,\rho_1}$ and $\lambda^{\sigma_2,\rho_2}$, bounding the traffic on two input ports of a node and directed to the same output port. Lemma 1 defines an operator $\oplus$ that combines the two traffic curves to compute a tight bound on the resulting aggregated traffic.

**Lemma 1.** *Let $\lambda^{\sigma_1,\rho_1}$ and $\lambda^{\sigma_2,\rho_2}$ bound the traffic on input ports (West, North or PE) directed to the same output port (East or South). Then the traffic on the output port is bounded by the following curve:*

$$(\lambda^{\sigma_1,\rho_1} \oplus \lambda^{\sigma_2,\rho_2})(t) = \min\left(t, \sigma_1 + \sigma_2 + \lfloor \rho_1 \cdot (t-1) \rfloor + \lfloor \rho_2 \cdot (t-1) \rfloor\right). \tag{5}$$

*Proof:* In any time window of length $t$, the number of packets transmitted on an output port cannot be greater than the traffic produced by the input ports; hence it holds:

$$(\lambda^{\sigma_1,\rho_1} \oplus \lambda^{\sigma_2,\rho_2})(t) \leq \sigma_1 + \sigma_2 + \lfloor \rho_1 \cdot (t-1) \rfloor + \lfloor \rho_2 \cdot (t-1) \rfloor.$$

Furthermore, since the number of packets cannot be larger than $t$, it also holds $(\lambda^{\sigma_1,\rho_1} \oplus \lambda^{\sigma_2,\rho_2})(t) \leq t$. Equation 5 then immediately follows. ■

Based on Equation 5, the operator $\oplus$ is both commutative and associative and we are essentially combining traffic flows. Hence, for any set $\mathcal{A}$ of traffic curves $\lambda^{\sigma,\rho}$, we write $\oplus\mathcal{A}$ to denote the aggregation of all curves in $\mathcal{A}$ based on the operator. We also write $\sigma(\mathcal{A})$ and $\rho(\mathcal{A})$ to denote the sum of the burstiness and rate parameters, respectively, for all traffic curves in $\mathcal{A}$. Note that based on Equation 5, this implies:

$$\oplus\mathcal{A}(t) = \min\left(t, \sigma(\mathcal{A}) + \sum_{\forall \lambda^{\sigma,\rho} \in \mathcal{A}} \lfloor \rho \cdot (t-1) \rfloor\right).$$

**Deriving Conflicting Flows $\Gamma_f^C$**: Having defined how to aggregate traffic curves for multiple flows, the next step is to define the set of **conflicting flows**, that is, those flows that block the injection of packets at the analyzed client. In particular, we consider a given flow $f \in F_{x,y}$ for the client at $(x, y)$, and define a set $\Gamma_f^C$ of traffic curves of other flows that conflict with $f$. Due to the complexity of the formal derivation, the full definition of $\Gamma_f^C$ is presented in Appendix B. Instead, here we provide intuition on how to derive such conflicting set. We show the set of interfering flows used to compute $\Gamma_f^C$ in Figure 7.

- First, we do not make any assumption on the arbitration used by client $(x, y)$ to decide which flow to serve. Hence,

$f$ can suffer self conflicts from any other flow in $F_{x,y}$ that is injected on the same port of $(x, y)$. For example, it may conflict on the South port if $f.x = x$, that is the destination of $f$ is on the same Y-ring as client $(x, y)$, or East otherwise.

- Second, we need to add to $\Gamma_f^C$ all flows generated by other clients that conflict with $f$. According to Table I, if $f$ injects packets to the East port, then it suffers conflicts from any flow ($W \rightarrow E$ or $W \rightarrow S$) arriving on the West port of $(x, y)$. If, instead, $f$ injects to the South port, it suffers conflicts from flows turning $W \rightarrow S$ at $(x, y)$, as well as $N \rightarrow S$ flows arriving on the North port of $(x, y)$ directed South.
- The set of flows going $N \rightarrow S$ is easy to determine, because flows are never deflected on a different Y-ring (it may deflect $N \rightarrow E$ but will never switch Y-rings). However, determining the set of flows on the West port is more involved, because traffic arriving on the North port of any node in the $y$-th X-ring can be deflected N→E on the ring itself if there is any traffic simultaneously turning W→S at the same node. Finally, if the optimization discussed in Section III is employed, where the router is modified to support PE→E + W→S routing at the cost of doubling the LUTs, the conflicting set $\Gamma_f^C$ can be modified to exclude W→S traffic in the case of East port injection.

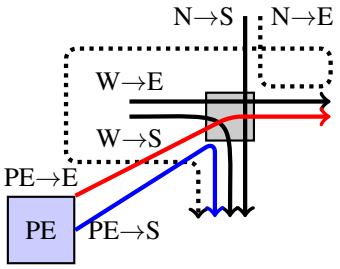



Fig. 7: Understanding interfering traffic flows at a client for determining the set of conflicting flows $\Gamma_f^C$. Dotted $N \rightarrow E$ is a deflected flow that will wrap around the X-ring and return at the $W$ port. The $PE \rightarrow E$ (red flow) will interfere with $W \rightarrow E$, and also $W \rightarrow S$ flow due to HopliteRT router limits. And, the $PE \rightarrow S$ flow (blue flow) will interfere with $N \rightarrow S$, $W \rightarrow S$, and the deflected $N \rightarrow E$ flows.

**Computing Bounds based on $\Gamma_f^C$:** Once the set of conflicting flows $\Gamma_f^C$ has been derived, we can now study the maximum delay suffered by client $(x, y)$ to inject a sequence of $k$ packets of flow $f$, under the assumption that $k \leq f.\sigma$. First, in the worst case, the sequence of packets can arrive at the client when the bucket has just been emptied; hence, in the worst case it will take a delay of $\lceil 1/f.\rho \rceil - 1$ before the bucket has one token. After such initial time, the packets can be further delayed by conflicting traffic in the network, which is bounded by $\oplus\Gamma_f^C$. Based on the properties of traffic curves, we can prove that the flow cannot be starved as long as the condition $\rho(\Gamma_f^C) < 1$ is satisfied. Intuitively, this means that the cumulative rate of conflicting flows is less than 1 packet/cycle; hence, eventually there will be available clock cycles when the flow can be injected on the NoC. The available rate of injection is thus $1 - \rho(\Gamma_f^C)$.

Assuming that the condition $\rho(\Gamma_f^C) < 1$ is satisfied, we then, as proved in Appendix C, show that:

- The first packet in the sequence can suffer a delay of at most $\lceil 1/f.\rho \rceil - 1 + T^s$ cycles, where:

$$T^s = \left\lceil \frac{\sigma(\Gamma_f^C)}{1 - \rho(\Gamma_f^C)} \right\rceil . \tag{6}$$

  Here, $T^s$ represents the delay caused by the burstiness of conflicting flows; it is proportional to the cumulative burst length of conflicting flows, and inversely proportional to the available injection rate $1 - \rho(\Gamma_f^C)$.
- For each successive packet in the sequence, the client suffers an addition delay of either $1/f.\rho$ or $1/\big(1-\rho(\Gamma_f^C)\big)$ cycles, whichever is higher. Here, the $1/f.\rho$ term represents the case where further packets are delayed by regulation, hence they are sent at the regulator rate $f.\rho$. The term $1/\big(1-\rho(\Gamma_f^C)\big)$ represents the case where packets are delayed by conflicting flows, hence they are sent at the available injection rate of $1 - \rho(\Gamma_f^C)$. In essence, we can prove that further packets in the sequence are delayed by either regulation or conflicting traffic, but not both.

This result, which is formally expressed by Theorem 2 in Appendix C, only holds for sequences of packets of length at most equal to the burst length $f.\sigma$ of the flow. The key intuition is that the burst length must be sufficient to allow the token bucket for $f$ to "fill up" while the flow is being blocked by conflicting NoC traffic. In the extreme case in which $f.\sigma = 1$, every packet in the sequence could suffer $\lceil 1/f.\rho \rceil - 1 + T^s$ delay, that is, it suffers delay due to both regulation and conflicting traffic. This reveals a fundamental trade-off for the burst length $\sigma$ assigned to each flow: if $\sigma$ is too small, then consecutive packets can be unduly delayed. However, a larger $\sigma$ increases the delay $T^s$ suffered by other clients. Finally, if $\rho(\Gamma_f^C) \geq 1$, then no delay bound can be produced as the flow might be starved indefinitely by conflicting traffic. If burst lengths can be chosen freely, an assignment of burst lengths could be computed with a distributed optimization problem to minimize the worst case latencies. Formulating and solving this optimization approach is left as future work.

## V. Evaluation

In this section, we show experimental validation of our bounds under various workloads, and packet flow configuration. We vary injection rates (in %), system sizes, traffic patterns, and real-time feature support and measure in-flight NoC latency and source queueing delays in the clients across our NoC. We compare the results to a baseline Hoplite NoC without real-time support. For in-flight latency tests, we evaluate synthetic traffic with 2K packets/client that exercises worst-case paths. We consider `LOCAL`, `RANDOM`, `TORNADO`, `TRANSPOSE`, and `ALLTO1` (everyone sends a packet to (0,0) client) traffic generators. For source queueing tests, we provide a tool to evaluate a user-supplied set of flows for feasibility and provide a bound where one can be proved.

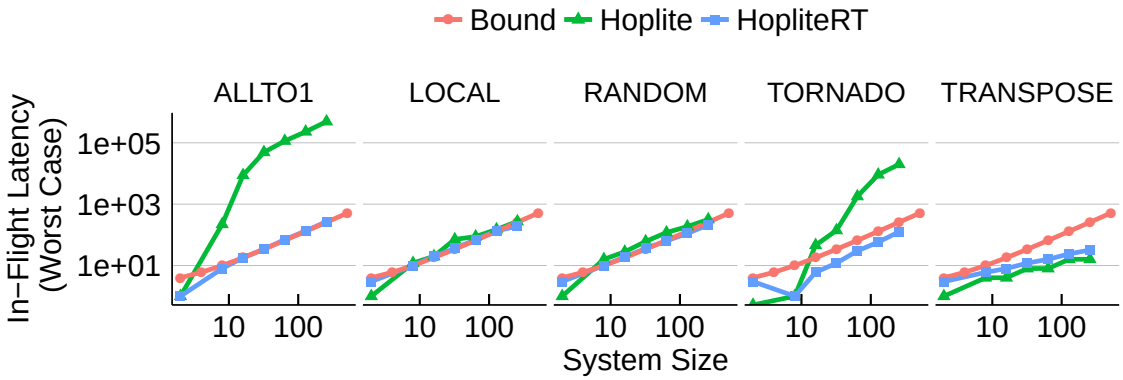



Fig. 8: Effect of Traffic Patterns on Worst-Case In-Flight Latency of the Workload at 100% injection rate. Worst-case analytical bounds (red) are easily violated by baseline Hoplite. With HopliteRT we are always within the bound, and deliver superior worst-case latency for ALLTO1, TORNADO, RANDOM, and LOCAL patterns. For TRANSPOSE, the persistent victimization of $N \rightarrow S$ packets causes a slightly longer worst-case latency.

**In-Flight NoC Latency Bounds**: In Figure 8, we show the effect of using HopliteRT over the baseline design when counting the worst-case latency suffered by the packet in the NoC. It is clear that for ALLTO1, TORNADO, RANDOM, and to some extend for LOCAL patterns, the worst-case latency with our extensions has been significantly improved. This is particularly true in the adversarial case where each client sends data to a single destination. This pattern is representative of situations where a limited resource like a DRAM interface must be shared across all clients in the system. Without HopliteRT, some client requests to a DRAM interface may never route to the DRAM interface unless other clients complete their requests[1]. For other traffic patterns, the benefit is less pronounced, and it gets slightly worse for TRANSPOSE. This is because W→S static priority victimizes $N \rightarrow S$ packets each time, which abundantly occur in other workloads. It is possible to improve fairness by using an extra priority bit and history information, but that would increase the cost of the NoC router. Finally, we show that our router never violates the predicted bounds that are calculated based on our analysis and these are better than the worst-case latencies observed in baseline Hoplite in most cases. Apart from the proofs detailed in the Appendix, this experimental validation supports a real-time developer in safely using these bounds during system design. In Figure 9, we take a closer look at a 256-client simulation and vary the injection rates from 0.1% to 100% for RANDOM workload. As expected, we observe that at low injection rates <10%, both networks deliver packets better than the bound. However, as network gets congested, the baseline Hoplite design delivers packets with increasing worst-case latency that exceeds the bounds. The HopliteRT design is always better than the bound at all injection rates. Our observed latencies are within 20% of the computed bound. The computed bound is consider tight, as it can be reached in some cases as in ALLTO1 and LOCAL.

[1]This behavior was demonstrated at the FCCM 2017 Demo Night where Jan Gray's GRVI-Phalanx [10] engine with 100s of RISC-V processors interconnected with Hoplite. Clients closer to the DRAM interface were effectively starved and never got service in a DRAM interface test.

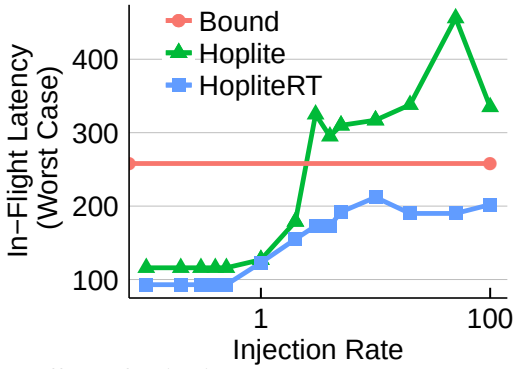



Fig. 9: Effect of Injection Rate on Worst-Case In-Flight Latency of the RANDOM Workload for 256 clients. At low injection rates, the NoC routing latencies are not very different, but as the NoC gets congested, HopliteRT starts to deliver improvements.

**Source-Queueing Bounds**: We now show the benefit of client injection regulation on source queueing delay. In these experiments, we use HopliteRT router in all comparisons and selectively enable regulation. This shows the need for regulation in addition to modification to the Hoplite router for delivering predictable, bounded routing latencies in the network. In this experiment, we set the offered rate $\rho$=$\frac{1}{m^2}$ ($m \times m$=size of NoC) and a burst $\sigma$=1. The traffic rate is scaled to $\frac{1}{m^2}$ to ensure feasible flows to the single destination client.

In Figure 10, we show the effect of using our regulator on source queueing delay for traffic with the ALLTO1 pattern. From this experiment, it is clear that simply using the HopliteRT router is insufficient and the source queueing times are large. When we add regulator hardware to the client, the waiting times drops dramatically by over four orders of magnitude. Our analytical bound tracks our observed behavior but there is a gap as it must assume pessimistic behavior from interfering clients in the calculations. We also consider RANDOM traffic pattern in Figure 11. Again, we observe better behavior with regulated traffic injection but the latencies are lower than the ALLTO1 case. While RANDOM traffic is less

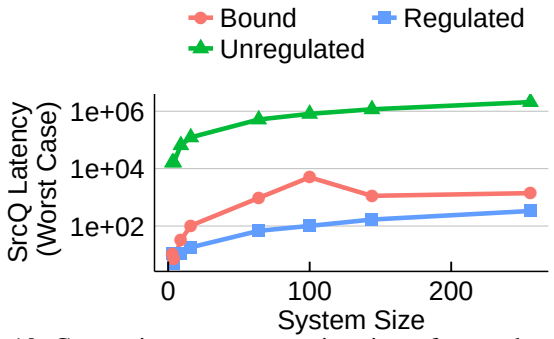



Fig. 10: Comparing source queueing times for regulated vs. unregulated HopliteRT NoCs as a function of system size for the ALLTO1 traffic pattern. Regulated traffic offers much improved waiting times at the clients.

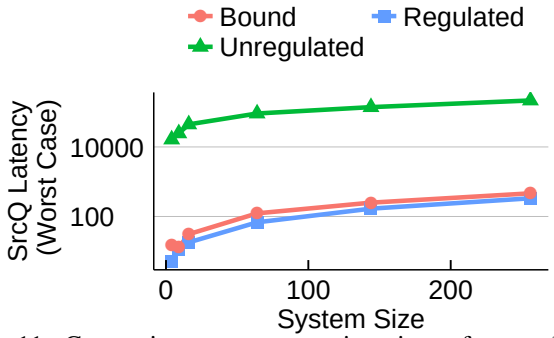



Fig. 11: Comparing source queueing times for regulated vs. unregulated HopliteRT NoCs as a function of system size for the RANDOM traffic pattern. Again, regulated traffic offers better latency behavior, but bounds are much lower than the ALLTO1 pattern.

adversarial than the ALLTO1 pattern, our bound still holds, and regulation is still up to two orders of magnitude lower latency. You may also note that the bound calculation is now significantly tighter than the ALLTO1 scenario as the traffic flows now interfere is less-adversarial manner.

Finally, we show a breakdown of the bounds for the PE South and East ports in Figure 12. The difference in bounds is because the $E$ port can accept more packets due to the DOR routing policy, and furthermore due to the limits of the routing configurations shown in Table I. Recall, we want to avoid doubling the LUT cost of the HopliteRT router, and intentionally disallow $PE \rightarrow E$ and $W \rightarrow S$ packets to route simultaneously. These have the effect of creating a difference in bounds for traffic injection along $S$ and $E$ ports as shown in the figure. The $E$ port suffers a higher waiting time as injection rate is increased as we disallow $E$ traffic in our LUT-constrained router.

## VI. Discussion and Related Work

Existing literature on real-time NoCs can be summarized in the following broad directions:

- Some research has built static routing tables for time-division NoCs such as those proposed in [11], [12]. However, this approach requires full knowledge of all communication flows, and is unsuitable for NoCs that needs to support both real-time flows requiring worst case guarantees and best effort flows where average case delay is important. [13] has different assumptions about the communication pattern, ranging from one-to-one restriction to none. It uses Time-Division Multiplexed (TDM) schedule to derive worst-case latency bound and splits traffic into multiple non-interfering flows. However, the bound limits the maximum rates of the flows depending on the number of non-interfering sets. The TDM schedule may exaggerate the required latency depending on the extent of interference. For HopliteRT, we only need to know the communication pattern and do not limit injection rates when flows are feasible.

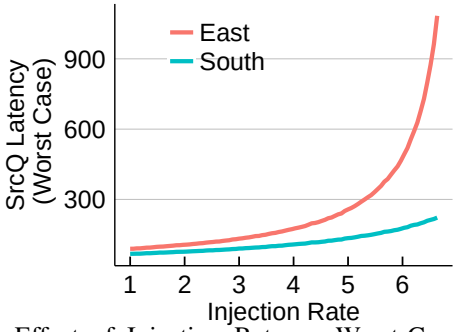



Fig. 12: Effect of Injection Rate on Worst-Case Source-Queueing Latency of the ALLTO1 workload for 16 clients. The $E$ port can accept more packets due to the DOR routing policy; and furthermore be blocked by our LUT-constrained HopliteRT router.

- Other work focuses on wormhole NoCs with virtual channels. The seminal work in [14] proposes priority-based networks, where each virtual channel corresponds to a different real-time priority, thus providing reduced latency for high-priority flows at the cost of low-priority ones. Recent work has extended the analysis to NoCs using credit-based flow control [15], as well as to round-robin, rather than priority-based arbitration [16]. However, these designs are expensive on FPGA, and require full knowledge of communication flows to derive tight latency bounds. In contrast, our approach rely on static modification of routing function as well as client regulators to bound latencies.
- Approaches such as Oldest-First [8] and Golden Flit [17] provide livelock freedom on deflection-routed networks similar to Hoplite, but are optimized for ASICs and use a richer mesh topology. On the other hand, minimally buffered deflection NoC [18] is suitable for FPGA and provides in-order delivery of flits eliminating the need for reassembly buffers. However, these reviewed approaches do not provide exact bounds on worst-case times.
- The use of regulators to bound the maximum network latency is well-known in the context of network calculus [19]. In [3], the authors show how to use Token Bucket regulators, similar to ours, to control traffic injection on the Kalray MPPA. Unlike HopliteRT, the Kalray NoC is source routed (client computes the complete path taken by the packet), and requires queuing at the client interface and within the NoC.

## VII. Conclusions

We present HopliteRT, an FPGA-based NoC with real-time support, that provides strong bounds on both (1) in-flight NoC routing latency due to deflection, and (2) the source-queuing waiting time at the source node. A 64b HopliteRT router implementation delivers approximately identical LUT-FF cost (2% less) compared to the original Hoplite router. We also add two counters to the client interface to provide a Token Bucket regulator for controlling packet injection in a manner that bounds source queueing delay. We show the in-flight NoC routing bound to be $(\Delta X + \Delta Y + (\Delta Y \times m) + 2)$, and the source-queueing bound to be $(\lceil \frac{1}{\rho_i} \rceil - 1) + T^s$ : $T^s = \lceil \frac{\sigma(\Gamma_f^C)}{1-\rho(\Gamma_f^C)} \rceil$. We test HopliteRT across various statistical datasets and show that (1) our analytical bounds are relatively tight for `RANDOM` workloads, and (2) our solution provides significantly better latency behavior for `ALLTO1` workload that models shared DRAM interface access.

Analysis tools are freely available to download at `https://git.uwaterloo.ca/watcag-public/hoplitert-bounds`.

**Acknowledgement:** This work was supported in part by NSERC and CMC Microsystems. The authors would like to thank Jan Gray for providing access to Hoplite RTL. 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.

## References

[1] N. Kapre et al. Hoplite: Building austere overlay NoCs for FPGAs. In *FPL*, 1–8 [2015].

[2] A. Olofsson, et al. Kickstarting high-performance energy-efficient manycore architectures with epiphany. *CoRR*, abs/1412.5538 [2014].

[3] M. Becker, et al. Partitioning and Analysis of the Network-on-Chip on a COTS Many-Core Platform. In *Proceedings of the IEEE RTAS* [2017].

[4] A. Kurdila, et al. Vision-based control of micro-air-vehicles: Progress and problems in estimation. In *43rd IEEE CDC*, volume 2, 1635–1642. IEEE [2004].

[5] H.-M. Huang, et al. Cyber-physical systems for real-time hybrid structural testing: a case study. In *Proc. 1st ACM/IEEE Int. Conf. cyber-physical Syst.*, 69–78. ACM [2010].

[6] M. K. Papamichael et al. CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs. In *the ACM/SIGDA international symposium*, 37. ACM Press, New York, New York, USA [2012].

[7] Y. Huan et al. FPGA optimized packet-switched NoC using split and merge primitives. In *FPT*, 47–52 [2012].

[8] T. Moscibroda et al. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th ISCA*, ISCA '09, 196–207. ACM [2009].

[9] A. Van Bemten et al. Network Calculus: A Comprehensive Guide [2016].

[10] J. Gray. GRVI-Phalanx: A Massively Parallel RISC-V FPGA Accelerator. In *Proc. 24th IEEE Symposium on FCCM*, 17–20. IEEE [2016].

[11] K. Goossens et al. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *47th DAC*, 306–311. IEEE [2010].

[12] N. Kapre. Marathon: Statically-scheduled conflict-free routing on fpga overlay nocs. In *2016 IEEE 24th FCCM (FCCM)*, 156–163 [2016].

[13] J. Mische et al. Guaranteed Service Independent of the Task Placement in NoCs with Torus Topology. In *Proc. 22Nd RTNS.*, RTNS '14, 151:160. ACM [2014].

[14] Z. Shi et al. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Second ACM/IEEE NOCS (nocs 2008)*, NOCS '08, 161–170. IEEE [2008].

[15] H. Kashif et al. Buffer Space Allocation for Real-Time Priority-Aware Networks. In *Proceedings of the IEEE RTAS*, 1–12. IEEE [2016].

[16] M. Panic, et al. Modeling High-Performance Wormhole NoCs for Critical Real-Time Embedded Systems. In *Proceedings of the IEEE RTAS* [2016].

[17] C. Fallin, et al. Chipper: A low-complexity bufferless deflection router. In *Proceedings of the 2011 IEEE 17th IS-HPCA*, HPCA '11, 144–155. IEEE Computer Society [2011].

[18] J. Mische, et al. Minimally buffered deflection routing with in-order delivery in a torus. *IEEE/ACM NOCS '17*, 1–8 [2017].

[19] J.-Y. Le Boudec et al. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg [2001].

## Appendix

### A. Derivation of In-Flight Latency Bounds

**Theorem 1.** *The in-flight latency of the HopliteRT is upper bounded by Equation 2*

*Proof:* As described in Section III, HopliteRT is engineered with a policy to prioritize the traffic turning from the X-ring to the Y-ring. This policy provides guarantees that a packet will be able to progress down on the Y-ring and cannot deflect more than once at every row. In the worst case, a packet can be deflected on every router during its journey on the Y-ring to the destination which is captured by Equation 2. ■

Equation 2 captures the worst-case bound of the in-flight latency with no assumption about the traffic, e.g., the communication patterns and rates are unknown. The bound can be improved farther by leveraging the knowledge about the traffic. In particular, we can reduce the number of deflection points on the Y-ring if we know that on specific X-rings there will be no conflicting traffic. To do this we include only the X-rings that can cause conflict. As a result, we can update Equation 2, to be as in Equation 7, to optimize the term $(\Delta Y \times m)$ to be $(V \times m)$, where $V \leq \Delta Y$ is the number of conflicting rows which is defined in Equation 8. The results of the optimized in-flight latency is beneficial especially in large networks. As shown in Figure 13, the optimized bound improves over the basic bound about 25% in the case of 16X16 network.

$$T^f = \underset{(y_1+1+DY)\%m}{\Delta X + \Delta Y + (V \times m) + 2} \tag{7}$$

$$V_{x_2,y_2}^{x_1,y_1} = \sum_{j=(y_1+1)\%m} (1) \mid \Gamma_{x_2,j}^{W2S} \neq \emptyset \tag{8}$$

### B. Derivation of Conflicting Flows

In this Appendix, we focus on formally deriving the set of conflicting traffic curves $\Gamma_f^C$ for a flow $f$ of client $(x, y)$, as discussed in Section IV-B. We first consider the set of conflicting flows of other clients, and then determine the set of conflicting flows of the same client $(x, y)$.

According to Table I, in order for a client to inject on the East port there must be no traffic on the West port. Similarly, to

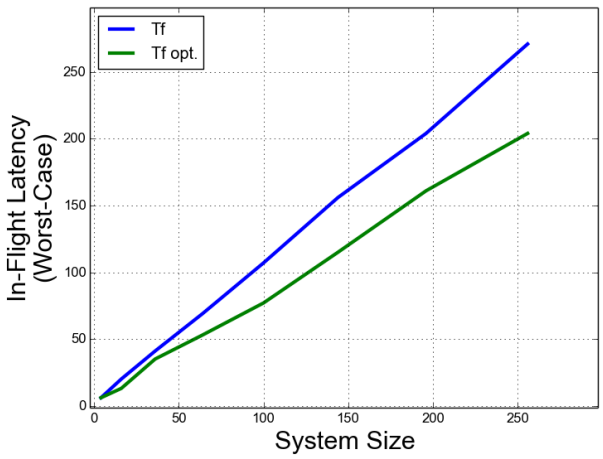



Fig. 13: The optimized bound versus the basic one on Worst-Case In-Flight Latency of RANDOM workload at 100% injection rate

inject on the the South port, there must be no North traffic and no West traffic turning South. Note that all North traffic goes South and does not turn East unless forcefully deflected due to a conflict on the South port. To know if a client is able to inject a packet, we need to know the total incoming traffic on each port. Equations (9 - 11) define the sets $\Gamma^{N2S}_{x,y}$, $\Gamma^{W2E}_{x,y}$, and $\Gamma^{W2S}_{x,y}$ of all traffic curves for source clients that have traffic on the North port of $(x, y)$ aiming South, on the West port aiming East, and on the West port aiming South respectively. $\Delta Y(f.y, j) \geq \Delta Y(y, j)$ and $\Delta X(f.x, i) > \Delta X(x, i)$ mean that the traffic passes through the router (X, Y) on the Y-ring or on the X-ring respectively.

$$\Gamma^{N2S}_{x,y} = \{\lambda^{f.\sigma,\ f.\rho}_{i,j} \mid \exists\ f \in F_{i,j}, \forall\ i, j \in [0, m) : (j \neq y) \wedge (f.x = x) \wedge \Delta Y(f.y, j) \geq \Delta Y(y, j)\} \quad (9)$$

$$\Gamma^{W2E}_{x,y} = \{\lambda^{f.\sigma,\ f.\rho}_{i,y} \mid \exists\ f \in F_{i,y}, \forall\ i \in [0, m) : (i \neq x) \wedge \Delta X(f.x, i) > \Delta X(x, i)\} \quad (10)$$

$$\Gamma^{W2S}_{x,y} = \{\lambda^{f.\sigma,\ f.\rho}_{i,y} \mid \exists\ f \in F_{i,y}, \forall\ i \in [0, m) : (i \neq x) \wedge (f.x = x)\} \quad (11)$$

Note that the presented sets do not consider the deflection effect. In the case of a conflict on the South port, the deflected traffic on the East port will cause extra pressure on all the clients in the X-ring. Since deflection does not increase the mount of traffic going South, the traffic on the North port of $(x, y)$ is simply $\lambda^{N2S}_{x,y} = \oplus\Gamma^{N2S}_{x,y}$. However, the same is not true for the traffic $\lambda^{W2E}_{x,y}$ and $\lambda^{W2S}_{x,y}$ on the West port of $(x, y)$. Consider a client $(i, y)$ on the same X-ring as $(x, y)$: when $\lambda^{W2S}_{i,y}$ and $\lambda^{N2S}_{i,y}$ conflict on the South port of $(i, y)$, some amount of traffic in $\lambda^{N2S}_{i,y}$ is deflected East and affects all the $m$ clients on the $y$-th X-ring. If the client under analysis $(x, y)$ wants to inject packets to the East, the deflected traffic at $(i, y)$ thus need to be added to the set $\Gamma^{W2E}_{x,y}$ of conflicting injected traffic flows.

It remains to determine the amount of traffic of $\lambda^{N2S}_{i,y}$ that can be deflected in the general case. Clearly, the amount of deflected packets in a window of length $t$ cannot be greater than $\lambda^{N2S}_{i,y}(t)$. Also, if there is no traffic turning West to South at $(i, y)$, that is if $\Gamma^{W2S}_{i,y} = \emptyset$, then no traffic can be deflected. In all other situations, we conservatively assume that all traffic produced by $\Gamma^{N2S}_{i,y}$ is deflected; even if the amount of traffic produced by $\Gamma^{W2S}_{i,y}$ is smaller than $\lambda^{N2S}_{i,y}$, deflected packets of $\lambda^{N2S}_{i,y}$ can travel around the X-ring and deflect further packets of $\lambda^{N2S}_{i,y}$ when turning W→S. Hence in the worst case, all N→S traffic will be deflected. Since the deflected traffic affects every client on the X-ring, we can thus define the set of aggregated deflected traffic $\Gamma^{dtot}_{y}$ that goes around in the X-ring as the union of the sets $\Gamma^{N2S}_{i,y}$ for any client $(i, y)$ such that there is traffic going W→S at that client:

$$\Gamma^{dtot}_{y}(t) = \cup_{\forall i \in [0,m)}\{\Gamma^{N2S}_{i,y} \mid \Gamma^{W2S}_{i,y} \neq \emptyset\}. \quad (12)$$

We can now compute the set of traffic curves $\Gamma^{CE}_{x,y}$ of other clients that conflict with injections on the East port at $(x, y)$, and the set of traffic curves $\Gamma^{CS}_{x,y}$ of other clients that conflict with injections on the South port at $(x, y)$. For $\Gamma^{CE}_{x,y}$, we combine the original traffic generated from other clients in the same X-ring, $\Gamma^{W2E}_{x,y}$ that pass to the East port and $\Gamma^{W2S}_{x,y}$ that turns south, plus the total deflected traffic on the X-ring:

$$\Gamma^{CE}_{x,y} = \Gamma^{W2E}_{x,y} \cup \Gamma^{W2S}_{x,y} \cup \Gamma^{dtot}_{y}. \quad (13)$$

For $\Gamma^{CS}_{x,y}$, we combine the original traffic generated from other clients in the same X-ring, $\Gamma^{W2S}_{x,y}$ that turns South, plus the traffic that arrives from the North port headed South, $\Gamma^{N2S}_{x,y}$:

$$\Gamma^{CS}_{x,y} = \Gamma^{W2S}_{x,y} \cup \Gamma^{N2S}_{x,y}. \quad (14)$$

Note that we do not consider the deflected traffic $\Gamma^{dtot}_{y}(t)$ as part of $\Gamma^{CS}_{x,y}(t)$, since among the deflected traffic, only the flows in $\Gamma^{N2S}_{x,y}(t)$ turn south at $(x, y)$. If the optimization discussed in Section III is employed, where the router is modified to support PE→E + W→S routing at the cost of doubling the LUTs, then the bounds can be improved. Since the W→S traffic does not affect injection on the East port anymore, the set of conflicting traffic on the East port is now equal to:

$$\Gamma^{CE}_{x,y} = \Gamma^{W2E}_{y} \cup (\Gamma^{dtot}_{y} \setminus \Gamma^{N2S}_{x,y}). \quad (15)$$

Finally, we determine the set of conflicting flows of the same client $(x, y)$. We make no assumption on the arbitration used by the client to decide which flow to serve. Hence, we consider a worst case where $f$ is the lowest priority flow: that is, if there is any other flow at $(x, y)$ ready to be injected to the same port (East or South) as $f$, then $f$ is blocked. Based on this assumption, we simply consider all other flows in $F_{x,y}$ as conflicting flows, similarly to traffic produced by other clients in the network. More in details, if $f.x = x$, then flow $f$ injects

to the South port at $(x, y)$ and the set of conflicting traffic curves is:

$$\Gamma_f^C = \Gamma_{x,y}^{CS} \cup \{\lambda_{x,y}^{f.\sigma,f.\rho} \mid \exists p \in F_{x,y}, p \neq f : p.x = x\}; \quad (16)$$

if instead $f.x \neq x$, then flow $f$ injects to the East port and the set of conflicting traffic curves is:

$$\Gamma_f^C = \Gamma_{x,y}^{CE} \cup \{\lambda_{x,y}^{f.\sigma,f.\rho} \mid \exists p \in F_{x,y}, p \neq f : p.x \neq x\}. \quad (17)$$

### C. Derivation of Delay Bounds

We now focus on formally deriving delay bounds for a sequence of $k \leq f.\sigma$ packets of flow $f$ injected by client $(x, y)$, as intuitively described in Section IV-B. In any window of time of length $t$, by definition there must be at least $t - \oplus\Gamma_f^C(t)$ free clock cycles, that is, clock cycles where the flow is not delayed by conflicting flows. Therefore, if the flow has sufficient tokens, it can inject up to $t - \oplus\Gamma_f^C(t)$ packets. The rest of the analysis proceeds as follows. First, in Lemma 2 we derive a condition under which the flow is not starved, that is, it will eventually receive free cycles. Assuming that such condition holds, in Lemma 3 we then show that:

$$t - \oplus\Gamma_f^C(t) \geq \max\left(0, \left\lfloor \left(t - (T^s + 1)\right) \cdot \left(1 - \rho(\Gamma_f^C)\right)\right\rfloor + 1\right), \quad (18)$$

where $T^s$ is defined as in Equation 6. This implies that the flow might receive no free cycles for $T^s$ clock cycles (it receives one for a window of length $T^s + 1$), but is then guaranteed to receive slots at a rate of $1 - \rho(\Gamma_f^C)$. Finally, note that the flow also cannot inject packets at a rate higher than the one of its regulator, $f.\rho$. In summary, as proven in Theorem 2, 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 $1/\left(1 - \rho(\Gamma_f^C)\right)$ cycles, whichever is higher.

**Lemma 2.** *Flow $f$ cannot suffer starvation if $\rho(\Gamma_f^C) < 1$.*

*Proof:* By expanding the expression for the guaranteed number of free injection cycles $t - \oplus\Gamma_f^C(t)$ we obtain:

$$\begin{aligned}
t - \oplus\Gamma_f^C(t) &= t - \min\left(t, \sigma(\Gamma_f^C) + \sum_{\forall \lambda^{\sigma,\rho} \in \Gamma_f^C} \lfloor \rho \cdot (t-1) \rfloor\right) \\
&= \max\left(0, t - \sigma(\Gamma_f^C) - \sum_{\forall \lambda^{\sigma,\rho} \in \Gamma_f^C} \lfloor \rho \cdot (t-1) \rfloor\right) \\
&\geq \max\left(0, t - \sigma(\Gamma_f^C) - \rho(\Gamma_f^C) \cdot (t-1)\right) \\
&= \max\left(0, t \cdot \left(1 - \rho(\Gamma_f^C)\right) - \left(\sigma(\Gamma_f^C) - \rho(\Gamma_f^C)\right)\right). \quad (19)
\end{aligned}$$

Now note that $\rho(\Gamma_f^C) < 1$ implies $1 - \rho(\Gamma_f^C) > 0$; hence, the number of guaranteed free slots increases with $t$, meaning that the flow cannot be starved. ∎

**Lemma 3.** *If $\rho(\Gamma_f^C) < 1$, then Equation 18 holds.*

*Proof:* The lemma follows directly by algebraic manipulation, where the last inequality is based on Equation 19.

$$\begin{aligned}
&\max\left(0, \left\lfloor \left(t - (T^s + 1)\right) \cdot \left(1 - \rho(\Gamma_f^C)\right)\right\rfloor + 1\right) \\
&\leq \max\left(0, \left(t - (T^s + 1)\right) \cdot \left(1 - \rho(\Gamma_f^C)\right) + 1\right) \\
&= \max\left(0, t \cdot \left(1 - \rho(\Gamma_f^C)\right) - \left((T^s + 1) \cdot \left(1 - \rho(\Gamma_f^C)\right) - 1\right)\right) \\
&= \max\left(0, t \cdot \left(1 - \rho(\Gamma_f^C)\right) - \left(\left(\left\lceil \frac{\sigma(\Gamma_f^C)}{1 - \rho(\Gamma_f^C)} \right\rceil + 1\right) \cdot \left(1 - \rho(\Gamma_f^C)\right) - 1\right)\right) \quad (20) \\
&\leq \max\left(0, t \cdot \left(1 - \rho(\Gamma_f^C)\right) - \left(\left(\frac{\sigma(\Gamma_f^C)}{1 - \rho(\Gamma_f^C)} + 1\right) \cdot \left(1 - \rho(\Gamma_f^C)\right) - 1\right)\right) \quad (21) \\
&= \max\left(0, t \cdot \left(1 - \rho(\Gamma_f^C)\right) - \left(\sigma(\Gamma_f^C) - \rho(\Gamma_f^C)\right)\right) \leq t - \oplus\Gamma_f^C(t).
\end{aligned}$$

∎

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

$$\lceil 1/f.\rho \rceil - 1 + T^s + \left\lceil (k-1) \cdot \max\left(\frac{1}{f.\rho}, \frac{1}{1 - \rho(\Gamma_f^C)}\right)\right\rceil. \quad (22)$$

*Proof:* In the worst case, the token bucket for $f$ can be initially empty for at most $\lceil 1/f.\rho \rceil - 1$ clock cycles. Afterwards, a new token is added to the bucket every $1/f.\rho$ cycles, at which point the next packet in the sequence becomes ready to be injected once the NoC port is free. Note that since $k \leq f.\sigma$, the times at which the first $k$ tokens are added, and thus the packets in the sequence become ready at the regulator, do not depend on the time at which the packets themselves are sent; this is because the bucket does not become full until the $k$-th token is added.

Now consider the effect of conflicting NoC traffic. Let $\lambda^{\text{free}}(t) = \max\left(0, \left\lfloor \left(t - (T^s + 1)\right) \cdot \left(1 - \rho(\Gamma_f^C)\right)\right\rfloor + 1\right)$, and consider any subsequence of $i$ packets out of the sequence of $k$ packets under analysis which are being delayed by NoC traffic. Since the time at which the packets become ready is fixed, the delay suffered by the last packet in the subsequence cannot be larger than both $\lceil (i-1) \cdot (1/f.\rho) \rceil$ and $\bar{t}$, where $\bar{t}$ is the minimum window length for which $\lambda^{\text{free}}(\bar{t}) = i$ (that is, the time that it takes for the NoC to have $i$ free cycles based on Lemma 3). Based on the expression for $\lambda^{\text{free}}$, it is then trivial to see that if $1/f.\rho \geq 1/\left(1 - \rho(\Gamma_f^C)\right)$, the worst case delay for the sequence is found when the first $(k-1)$ packets are sent as soon as they become ready at the regulator, while the last packet suffers NoC delay of $T^s$; while if $1/f.\rho \leq 1/\left(1 - \rho(\Gamma_f^C)\right)$, the worst case is found where all $k$ packets are delayed by NoC traffic rather than regulation. Combining the two cases yields Equation 22. ∎

Theorem 2 only holds for sequences of packets of length at most equal to the burst length of the flow. If the sequence is longer than the burst length, then the token buffer might become full during a window of length $T^s$ when the NoC port is blocked, at which point the time when further tokens are added is delayed based on when packets in the sequence are sent. This in turn causes extra delay.