Reduced Worst-Case Communication Latency Using Single-Cycle Multi-Hop Traversal Network-on-Chip

Peng Chen, Weichen Liu, Hui Chen, Shiqing Li, Mengquan Li, Lei Yang, and Nan Guan

***Abstract*—The communication latency in traditional Network-on-Chip (NoC) with hop-by-hop traversal is inherently restricted by the distance between source-destination communicating pairs. SMART, as one of the dynamically reconfigurable NoC architectures, enables the new feature of single-cycle long-distance communication by building a direct bypass path between distant cores dynamically at runtime. With the increasing of the number of integrated cores in multi/many-core systems, SMART has been deemed a promising communication backbone in such systems. However, SMART is generally optimized for average-case performance for best-effort traffics, not offering real-time guaranteed services for real-time traffics, and thus SMART often shows extremely poor real-time performance (e.g., schedulability).**

To make SMART latency-predictable for real-time traffics, by combining with the single-cycle bypass forwarding technique, in this paper, we firstly propose a priority-preemptive scheduling to allow contending packets to be arbitrated according to predefined priorities. Based on the priority-based scheduling, for the real-time packet flows with given flow mapping and predefined priorities, we then propose a real-time communication analysis model, by considering shared virtual channels (or priority levels) and arbitrary-deadline real-time packet flows, to predict the *worst-case communication latency* and validate the schedulability. Through theoretical and experimental comparison, the *worst-case communication latency* of the analyzed packet flows is reduced significantly compared with that of the traditional priority-preemptive NoCs with hop-by-hop traversal and the original distance-based SMART, thus improving the schedulability.

***Index Terms*—SHARP NoC, real-time communication analysis, priority/VC share, arbitrary deadline, priority-preemptive communication scheduling, worst-case end-to-end latency.**

I. Introduction

THE network-on-chip (NoC) architecture is becoming an increasingly important communication backbone for Multi-Processor System-on-Chip (MPSoC) platform, due to its high communication efficiency and good scalability. In particular, the real-time systems make a significant portion of embedded multi-core systems based on the MPSoC platform. The real-time constraints of real-time applications pose challenges to the design of NoC. For the real-time applications that are mapped on such NoC-based MPSoC, real-time communication guarantee is an essential requirement; that is, besides the *logical* requirement (i.e., completion of transmission), the *temporal* requirement should also be met (i.e., all of the real-time packets must be delivered to the destination before the specified deadline). Otherwise, the real-time applications may have significantly degraded performance, even suffer catastrophic damage. To ensure the timely delivery of packets, we should firstly calculate a safe and tight worst-case end-to-end latency of each packet flow at design time under the current resource constraints. And then, by employing the upper bound of the analytical latency, an optimal schedulable order (i.e., priority) can be found, and the resource assignment can be well controlled to avoid over-provisioning. To this end, many research works [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] are conducted continuously on worst-case end-to-end latency to get a tighter and safe latency upper bound in traditional priority-preemptive wormhole NoCs. However, the latency lower bound (namely zero-load latency C_i without any contentions) is inherently determined by the number of hops between the source and destination in traditional NoCs with hop-by-hop traversal. There is a chance for hard real-time applications that deadlines D_i would be violated even under zero load conditions (formally $C_i > D_i$, i.e., if a flow is somehow assigned a long-distance route, the zero-load latency C_i would be potentially more than the preset deadline D_i), especially for these applications with very tight deadlines; otherwise, we need to reallocate more resources to guarantee the schedulability (i.e., all of the flows are schedulable). Besides, the state-of-the-art works [11], [12] didn't consider the priority/VCs share and the general arbitrary-deadline flows.

Different from traditional hop-by-hop traversal NoCs, an advanced NoC design is SMART [13], [14], [15], [16], [17], *Single-cycle Multi-hop Asynchronous Repeated Traversal*, that can enable single-cycle multi-hop bypass transmission, removing the hop count dependence between the source and destination of the communication pairs, especially beneficial for multi/many-core systems. Despite the benefits brought by SMART, however, only the distance-based arbitrations are supported, which cannot provide real-time guaranteed services especially for real-time packet scheduling by predefined priorities, and thus the SMART shows poor real-time performance. The distance-based SMART platform thus cannot be directly applicable for real-time applications. And even if it supports priority-based scheduling, since the analyzed packet sometimes bypasses some routers and is stopped at intermediate routers where contention occurs, SMART introduces non-determinism and complicates the real-time analysis.

To make SMART latency-predictable for real-time systems, in this paper, we firstly propose a priority-preemptive com-

P. Chen and M. Li are with the College of Computer Science, Chongqing University, Chongqing, China, and also with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: (chnp616@gmail.com).

W. Liu, H. Chen, and S. Li are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore.

L. Yang is with the Department of Electrical Computer Engineering at the University of New Mexico, NM, USA.

N. Guan is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China.

munication scheduling based on the state-of-the-art evolved SMART NoC [16], called SHARP. Based on the proposed scheduling policy, the real-time packets in conflicting routers are locally arbitrated according to predefined priorities, and the real-time guaranteed services can be provided. The main contributions of this paper are summarized as follows:

- We first propose a priority-preemptive communication scheduling policy in SMART NoCs, which supports differentiated scheduling based on predefined priorities.
- We present a real-time communication analysis model for constrained-deadline packet flows (deadlines are not more than periods) by considering shared priority levels so that the end-to-end latency is predictable.
- We extend the real-time analysis to the arbitrary-deadline packet flows (deadlines can be $(<, =, >)$ periods).
- Through theoretical and experimental comparison, our proposed approach outperforms the traditional counterparts and the distance-based evolved SMART.

We would like to acknowledge that we rewrite some of the classical real-time formulas of the existing works (e.g., [10], [11], [12]), but the extension of analysis to SMART and arbitrary-deadline flows is nontrivial due to more complex communication scenarios and non-determinism. Besides, these state-of-the-art existing works didn't consider the priority/VCs share and arbitrary-deadline flows. To the best of our knowledge, this is the first work introducing the evolved SMART NoC to real-time many-core systems. The proposed latency bounds may also suffer some sources of pessimism, and we will continue to refine the analysis tightness in future work.

The remainder of this paper is organized as follows: Section II provides the background of SMART/SHARP. Section III describes the priority-preemptive communication scheduling policy used in our paper. Section IV describes the problem definition and presents the corresponding system model. Section V proposes the real-time communication analysis model regarding the estimation of worst-case end-to-end latency. Section VI experimentally compares the latency bounds with that of the traditional counterparts and original distance-based evolved SMART. Section VII finally concludes this paper.

II. Background

In this section, we first introduce the original SMART NoC [13], [14], [15]. Recently, there are many new works (e.g., [18], [16], [17]) improving SMART or based on SMART (e.g., [19], [20], [21], [22]). The evolved SMART NoCs including SSR-Net [18] and SHARP [16] are introduced later.

A. Overview of SMART NoC

To achieve single-cycle multi-hop bypass transmission in SMART [13], *Single-cycle Multi-hop Asynchronous Repeated Traversal*, a novel low-swing clockless repeated link circuit is embedded in the router crossbar at each router to replace conventional clocked link drivers, together with additional multiplexers and asynchronous repeaters. Different from the traditional router, *Switch Allocation* (SA) of the SMART router is split into two steps, referred to as the *Local Switch Allocation* (SA-L) and *Global Switch Allocation* (SA-G). The start router firstly selects a winner from all of the buffered flits that have applied a request for the target output port at the SA-L stage. After that, a(n) *SMART-hop Setup Request* (SSR) request that carries route information of *SA-L* winner is broadcasted along a predefined dimension-order route (e.g., XY route) to the downstream routers up to HPC_{max} hops (where HPC_{max} refers to the maximum number of hops can be reached in a clock cycle [13]) via dedicated control links to establish a bypass data path (namely *SSR* stage). During the next SA-G stage, each *SSR* recipient conducts the arbitration in parallel by following one of the two distance-based arbitrations namely *local priority* and *bypass priority*. When there are multiple contending *SSR* requests, *local priority* policy authorizes the priority to the *SSR* request that is from the nearest start router. While for *bypass priority* policy, the *SSR* that is from the farthest start router wins *SA-G* arbitration. After that, the single-cycle bypass data path is established by setting the values of control signals at each data router where *SSR* wins along the downstream path. Finally, the corresponding flits of *SA-G* winner traverse all of the reserved intermediate routers in the *Switch Traversal* (ST) + *Link Traversal* (LT) stages until they arrive at the stopping routers or destination.

At the data transmission phase, due to the VC selection failure or some other situations, body/tail flits may be prematurely stopped at intermediate routers. To tackle this problem [13], a VC is reserved by head flit at all of the intermediate routers beforehand at the path setup phase, ensuring that the body/tail flits can identify which VC at which they can be buffered when a failure occurs at the downstream routers. And the VC can be reserved and freed at all the routers between the start router and the stop router within the same cycle.

B. Evolved SMART NoCs

Compared with traditional hop-by-hop traversal NoCs, SMART indeed reduces the average communication latency by a maximum of $8\times$ [13] via the preset single-cycle multi-hop bypass path. However, SMART incorporates a large number of dedicated control links to broadcast *SSR* requests, and as a result, such design brings large overhead in wire, energy, and area. An evolved SMART (referred to as SSR-Net) is proposed in [18] to mitigate these problems. In an SSR-Net, *SSR*s are transmitted using a dedicated auxiliary network (called *control network*), which is in parallel to *data network*. Before an *SSR* is sent in the control network, a corresponding *pre-SSR* request is broadcasted in advance to establish the *SSR* routing path at the control network for *SSR* transmission by setting the control signals. According to the arbitration decision of *pre-SSR* requests, only the *SSR* with the highest priority is forwarded to downstream data routers through the control network. Thus, the overlapping control links (e.g., SSR link) in SMART are replaced with the shared control network. Besides, the data router only arbitrates the *SSR* with the highest priority from the upstream data router and local request, which also simplifies the switch allocator design of the data router. Since the width of *pre-SSR* wires is much smaller than *SSR* wires, SSR-Net requires lower control overhead than SMART.

However, the *pre-SSR* arbitration of SSR-Net is the same way as the *SSR* arbitration of SMART. Each router in SSR-Net

still independently conducts the SA-G arbitration in parallel, without considering the arbitration results of the upstream routers. Thus, the *pre-SSR* stage of SSR-Net also suffers the quadratic complexity (i.e., among up to $HPC_{max}(2HPC_{max}-1)$ requests). Besides, as illustrated in [13] and like SMART, the *pre-SSR* stage of SSR-Net additionally suffers the *false negatives* phenomenon. Such a phenomenon is the case where a multi-hop path is built in a router, but no incoming flits arrive, which results in throughput loss. To avoid the quadratic complexity arbitration and *false negatives* phenomenon, the authors in [16] recently propose further evolved SMART (referred to as SHARP). In a SHARP, instead of arbitrating among up to $HPC_{max}(2HPC_{max}-1)$ *SSR*s and *pre-SSR*s in SMART and SSR-Net, SHARP sequentially arbitrates the incoming *SSR*s router by router up to HPC_{max}. Only the winning *SSR* will be forwarded to the next router to build the single-cycle multi-hop path. Meanwhile, the authors identify that the SSR arbitration at the input port is redundant since the same arbitration result is acquired in the output port of the upstream adjacent router. By eliminating the quadratic complexity and redundant arbitration, the SA-G stage in SHARP is significantly simplified and consumes much less time than that of SMART and SSR-Net. However, eliminating the quadratic complexity and avoiding the *false negatives* will make HPC_{max} slightly reduced (i.e., HPC_{max} is slightly reduced from 8 to 6 in SMART and SHARP using 1mm per-hop and 1ns clock) due to the sequential *SSR* arbitration. Since the maximum HPC_{max}-hop path is actually difficult to build with increasing traffic, the slight HPC_{max} reduction can be accepted compared with the significant benefits (i.e., quadratic complexity elimination, throughput improvement). In the rest of the paper, we conduct our work based on the latest evolved SMART, called SHARP.

III. Priority-Preemptive Scheduling Policy

To fully support user-defined priority scheduling for real-time traffics in SHARP, we strategically present the priority-preemptive communication scheduling policy together with the discussion related to possible implementation. Instead of arbitrating the contending packets by distance, our proposed scheduling policy allows to arbitrate them according to predefined priorities, offering more degree of latency predictability.

A. Priority-based Scheduling

Before packet transmission in SHARP NoC [16], the incoming packets will experience two rounds of arbitration: *local switch allocation (SA-L)* that uses any arbiters according to quality-of-service (QoS) requirements at the source/start router and *global switch allocation (SA-G)* that is simply based on the distance at the downstream routers. To make SHARP latency-predictable and provide real-time guaranteed service, we alternatively achieve the arbitration with a fixed-priority arbitration module in this paper, in which *SA-L* (resp. *SSR*) stage can conduct the local (resp. global) arbitration by predefined priorities. To maintain low latency, the corresponding priority module should be kept simple and easy to implement, instead of the complex ones. Therefore, a low-latency arbitration module can be employed (e.g., the comparator with a binary comparison tree that is based on the fast adder circuits [23]).

Ideally, we would like to have as many priority levels as packet flows. However, such a way would increase the number of bits of *SSR* signal and energy consumption significantly, but conducting the priority-based arbitration would be also complex. Some different packet flows can potentially share the virtual channels (VCs) and have the same priority. As for the same-priority packets, the first-in-first-out (FIFO) mechanism is adopted. Therefore, to reduce the arbitration complexity and control overhead, a large number of packet flows are divided into a small number of priority levels. And, in credit-based NoC, the buffer turnaround time t_b is about 6 cycles [24]. To cover such time, multiple VCs (i.e., 6 VCs) per priority level is achieved. Hence, the number of priority levels is set as $\lceil n_{vc}/t_b \rceil$ (i.e., n_{vc} is the total number of VCs of each input port); each VC has specified a distinct priority and been exclusively assigned to a packet during a time interval. A group of VCs with the same priority are shared by multiple same-priority flows. The priority information is encoded in *SSR* with width of $\lceil \log_2(n_{vc}/t_b) \rceil$ bits. During the *SA-L* and *SSR* arbitration stages, note that multiple requests with the same priority may release simultaneously for the same output port or input port. For the packets belonging to the same priority, the input port will choose the packet that arrives earliest. As for the packets requesting the same output port, like the original SHARP, the arbiter (e.g., output port arbiter) authorizes the priority according to the incoming direction: *straight* $>$ *left-turn* $>$ *right-turn*.

B. Preemption Behavior

Safety/time-critical applications need not only account for the completion and correctness of the packet transmission but also the timely delivery before the specified deadline. In original SMART [14], [13], Tushar et al. mentioned that the virtual cut-through flow control [25] can be adopted for simpler router design and correctness, applicable for non-real-time best-effort traffics. With virtual cut-through, the communication resources (e.g., link channel) are allocated in units of packets (i.e., packet granularity). It is indeed beneficial for best-effort traffics, but it cannot work well for real-time traffics. Specifically, the higher-priority packets may suffer the blocking latency from an entire packet with lower priority that is being transmitted in the worst case. And then these higher-priority packets may not finish transmission before their respective deadlines at the destination, thus degrading the real-time performance in terms of schedulability. Furthermore, the buffer storage cost is relatively high (i.e., the buffer depth is at least equal to the maximum number of flits of a packet), and even the buffers are not used efficiently. By contrast, wormhole flow control [25] is widely used with more flexible router design in traditional hop-by-hop traversal NoCs, in which the communication resources are allocated in units of flits (i.e., flit granularity) and a small number of flit buffers is required at each router, especially beneficial for real-time traffics [2].

In SHARP NoC [16], the global switch allocation (SA-G) consists of three steps: input arbitration, output arbitration,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3015440, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

and configuration. In the following, we will detail the possible implementation of priority-preemptive wormhole mainly from the input arbitration. Before introducing the updated input arbitration, to guarantee the correctness of packet transmission especially under the packet interleaving situations, let ($source\_id$, $packet\_id$) be used as the key value to uniquely identify the VC at the VC-to-source (packet) mapping table. This table is checked sequentially from the source to the next stopping router when establishing the bypass path [16]. Besides prioritizing the incoming *SSR* and local request based on priority and FIFO policy, the input arbitration also conducts the *blocking tests*, i.e., guarantee flit ordering and unique VC-to-source (packet) mapping, illustrated as follows:

- First, with the priority and FIFO policy, the arbiter aims to select a winner from the incoming *SSR* and a local request that won the SA-L stage for the input port.
- Second, when there are prematurely stopped flits from the same packet in a router, the incoming flit will be buffered at that router to ensure flit ordering at the destination if free flit buffers exist, else buffered at the previous router.
- Third, the arbiter also needs to check the availability of free buffer/VC at the next-hop router for a given *SSR*: (i) If the incoming flit is a head flit, the intermediate downstream routers are checked for a free VC with corresponding priority to allocate to this head flit. To ensure the body/tail flits have a VC, the head flit reserves a VC at all routers from the start router to its stop router. Distinct from the VC selection at the stop router in SMART [13], the VC selection of our wormhole in SHARP [16] is conducted before head flit transmission. This is because, in SMART [13] with *SSR* broadcast, the presumed next stop router is not ensured and multiple flits from different start routers might choose the same VCid, but this case will not occur in SHARP [16] due to sequential *SSR* propagation. (ii) While if the incoming flit is a body or tail flit, the flit buffer availability of the next downstream router is checked. If there are free flit buffers, the incoming flit will be buffered at the current router, else the flit is stopped at the previous router. The *credit* signal is still between neighbors, and the flit buffers/VCs are reserved and freed at all the routers from the start router to the stop router within the same cycle. Actually, in SHARP [16], the *SSR* (resp. *credit*) signal is sequentially propagated (not broadcasted) from start (resp. current stop) router to the downstream stop (resp. upstream start) router, the total control time between start and stop routers is within 1 cycle [16], i.e., 1ns in a technology of TSMC 45nm and 1GHz. Due to the sequential arbitration of SSR, the experimental synthesis results in SHARP [16] show that the HPC_{max} value will be slightly reduced from 8 to 6 hops. Compared with the original SMART [13], SHARP avoids the false negative phenomenon, which achieves higher communication performance. Thus, a slightly lower HPC_{max} value can be acceptable.
- Finally, the arbiter checks to see if the current router is the destination by using the $eject\_flag$ field. If the current router is the destination, the incoming flit will bypass the router pipeline of the destination and be forwarded to the PE out of the port indicated by $eject\_port\_id$ field.

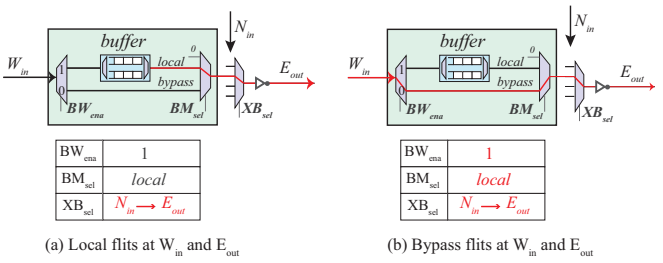


Fig. 1. Two preemption examples of control signals resetting. (a) Local (b) Bypass flits are preempted by the flits from another input port N_{in}.

While SA-L arbitrates buffered flits by priority and FIFO policy; the output arbitration is conducted by priority and direction, and configuration sets up the multi-hop path. Together, these aforementioned updated operations constitute the wormhole-based SHARP. According to the flit-level wormhole-based communication scheduling, the transmitting packet with low priority can be potentially preempted by higher-priority flows. Considering the flits that are being transmitted on shared resources, these flits are from either local buffered flits or bypass flits, shown in Fig. 1. Since the E_{out} and subsequent link are shared resources, to reduce potential blocking from lower-priority packets that are being transmitted, such resources allow being reassigned to the incoming flits with higher priority. In Fig. 1(a), suppose the local flits from W_{in} are holding the output port E_{out} and subsequent link. During their transmission, the requestor with higher priority from another input port N_{in} applies to access to the shared resources. In this case, the local flits will be preempted by incoming packets from N_{in}. The established path of local flits is disabled by resetting corresponding values of control signals, shown in Fig. 1(a). Similarly, suppose the bypass flits from W_{in} are being transmitted in Fig. 1(b). The control signal settings of the original transmitting flow are shown in red color at the table, shown in Fig. 1.

C. Implementation Consideration

Our proposed priority-preemptive wormhole-based communication scheduling mainly replaces the *distance* field with *priority* to encode *SSR*. To support the priority-preemptive wormhole technique, the arbitration module of the original router requires some modifications (i.e., replace the distance-based arbiter with the priority- and FIFO-based one). In the VC-to-source mapping table at the router input port, coupled with $source\_id$, $packet\_id$ is also included to uniquely identify which VC for each packet. To check the availability of flit buffers, the credits are tracked in detail. Aside from these main modifications, wormhole-based SHARP essentially follows the same pipeline as virtual cut-through based SHARP [16]. We would like to acknowledge that this additional modification may incur higher control costs. To realize the real-time guarantee in time-critical scenarios, this additional overhead is necessary. In this work, we mainly focus on the strategic design of real-time support with SHARP techniques and did

not restrict the specific realizations. Besides the scope of the real-time theoretical analysis and the wormhole strategic design, the complete exploration and evaluation of the wormhole implementation will be studied in our future work. Distinct from virtual cut-through, coupled with wormhole technique in SHARP, when the hop count between source-destination communication pairs is larger than HPC_{max}, the entire packet does not need to be fully buffered at the HPC_{max}-limited router. Instead, the flits of the packets are mainly buffered at source/start/destination routers, intermediate routers, and HPC_{max}-limited routers along the dimension-order route (e.g., XY route), thus the router buffer size does not need to be large enough to store the entire packet.

IV. System Model

In Section III, a priority-preemptive communication scheduling policy in SHARP is proposed to support differentiated scheduling according to predefined priorities. Combined with the communication advantage of SHARP and the proposed scheduling policy, priority-driven real-time applications will benefit a lot from that platform. In a wide range of hard real-time systems, especially for safety/time-critical systems, the real-time communication guarantee is an essential requirement. Specifically, for a given real-time application running in such SHARP NoC-based multi/many-core systems, it needs several components with computation/storage requirements (e.g., processors, memory, and GPU), namely processing element (PE), to run. To improve parallel performance, the cooperating tasks are usually mapped to different PEs. Therefore, real-time packet flows are required among such components to exchange data, and analyzing *worst-case end-to-end latency (WCEL)* of flows at design time is of fundamental importance to validate the schedulability, i.e., whether all of the real-time packet flows (*flow* for short henceforth) are delivered to destinations before their deadlines or not in the worst case. A generated instance of a *flow* at runtime is denoted as *packet*, similar to the relationship between *job* and *task* in real-time task scheduling (i.e., a *task* instance is usually called *job*).

TABLE I
Notations Used in Real-time Communication Analysis.

Notation	Definition
PE_x, R_y, ℓ_z	Processing element, SHARP Router and unidirectional link
buff	Buffer depth for each virtual channel
t_r	The router-stage latency to set up data path
t_w	The time to forward a flit to next stopping router
F	The set of packet flows in a SHARP NoC
f_i	A real-time packet flow
$f_{i,\lambda}$	The λ^{th} packet of f_i
L_i	Packet length in flits of f_i
D_i	Relative deadline of f_i
T_i	The minimum interval time between two successive arriving packets of f_i
P_i	The priority of f_i
J_i	The release jitter of f_i
$path_i$	Traversal path of f_i in traditional NoC
$path'_i$	Transformed path of f_i in SHARP NoC
sl_{ij}	The shared links between f_i and f_j along $path_i$
sl'_{ij}	The shared links between f_i and f_j along $path'_i$
C_i	Zero-load latency of $path_i$
C'_i	Zero-load latency of transformed $path'_i$
$S_D(f_i)$	Higher-priority flows that directly contend with f_i
$S_I(f_i)$	Higher-priority flows that indirectly contend with f_i
$S_I^{dn_j}(f_i)$	Higher-priority flows that indirectly contend with f_i via $f_j \in S_D(f_i)$ at downstream position
I_{ji}^{I}	The indirect interference suffered by f_i from f_j
I_{ji}^{dn}	The downstream indirect interference suffered by f_i from downstream direct flow f_j
$S_{SD}(f_i)$	The set of same-priority flows that directly contend with f_i
$S_{SI}(f_i)$	The set of same-priority flows that indirectly contend with f_i
$S_{SI}^{dn_j}(f_i)$	The set of same-priority flows that indirectly contend with f_i via $f_j \in S_{SD}(f_i)$ at downstream position
SB_i	The queue blocking from same-priority flows
HI_i	The interference from higher-priority flows
dib_{si}	Downstream indirect blocking from same-priority flow $f_s \in S_{SD}(f_i)$
dii_{hi}	Downstream indirect interference from higher-priority flow $f_h \in S_D(f_i)$
HI_{hi}	The interference from $f_h \in S_D(f_i)$
R_i	Worst-case end-to-end latency (WCEL) of f_i
$R_i(\lambda)$	Busy period upper bound of f_i

A. System Model

TABLE I gives the notations and their definitions used in this paper. A given set of periodic or sporadic flows F in a SHARP consists of the flows f_1, f_2, ..., and f_n, where a flow f_i is specified as (L_i, D_i, T_i, P_i, J_i). L_i is the complete packet length in flits incorporating all the necessary parts (i.e., head flit, body flit, and tail flit). D_i denotes the relative deadline. T_i refers to the minimum interval time between two successive arriving packets. P_i denotes the priority of flow f_i, with all the generated packets from f_i share the same priority. A smaller integer of P_i represents a higher priority. That is to say, if $P_i < P_j$, f_i has a higher priority than f_j. J_i is the release jitter that denotes the maximum deviation of packets released from its period. Note that, the flow mapping (i.e., task allocation) and the corresponding priorities are given. The priority assignment can be determined by static-priority scheduling strategy, i.e., Rate Monotonic (RM), and Deadline Monotonic (DM) scheduling [26] or other feasible ways ([27], [21]). Different system model (e.g., constrained-deadline task system) has different appropriate scheduling and task allocation strategy. The way to choose the priority assignment in scheduling is based on the goal of maximizing the percentage of schedulable flows under current resource constraints. The complete route of f_i composes of two *PE-to-Router* links and multiple *Router-to-Router* links. For simplicity, we can only consider the *Router-to-Router* path since the times in *PE-to-Router* and *Router-to-PE* links are constants. Let $path_i$ denote the *Router-to-Router* path from the source router to the destination router, namely $path_i = \{\ell_1, \ell_2, ..., \ell_m\}$. The absolute value $|path_i|$ also represents the number of links along the path $path_i$, and the links are numbered in order. Let sl_{ij} denote the shared links between flows f_i and f_j, where $|sl_{ij}|$ also denotes the number of shared links along their paths, then $sl_{ij} = path_i \cap path_j$. As mentioned in Section III-B, the wormhole switching, and XY-routing are employed due to practical consideration for real-time traffics. Note that, our real-time analytical model is applicable for SHARP-1D and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3015440, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

SHARP-2D versions, and not limited to one of them. Which version to be adopted depends on the performance you desire.

According to the aforementioned system model, assume the zero-load latency C_i of an analyzed flow f_i is the worst-case communication latency with the consideration of contention-free traversal and the limitation of the maximum number of bypass hops HPC_{max}. Since the path setup time for head flit and link latency for a body flit are constants, the value C_i is determined by the effective number of hops $\left\lceil \frac{|path_i|}{HPC_{max}} \right\rceil$ and packet length L_i. Therefore, the zero-load latency C_i (*Router-to-Router* path) in wormhole SHARP is formulated as:

$$C_i = (t_r + t_w) \cdot \left\lceil \frac{|path_i|}{HPC_{max}} \right\rceil + t_w \cdot (L_i - 1). \quad (1)$$

While the zero-load latency C_i (*Router-to-Router*) in traditional priority-preemptive wormhole NoCs is formulated as:

$$C_i = (t_r + t_w) \cdot |path_i| + t_w \cdot (L_i - 1). \quad (2)$$

Where t_r is the router-stage latency to set up data path, and t_w refers to the wire latency between two logically adjacent stopping routers. If the worst-case end-to-end latency R_i (that is derived by adding the summation of blocking/interference latency to the basic zero-load latency) meets the condition $R_i \leq D_i$, any generated packet from an analyzed flow f_i will never miss its deadline at runtime under the worst communication case, which is considered *schedulable*.

B. Communication Flow Graph Transformation

Despite the greatly enhanced communication efficiency of SHARP, it also introduces increased non-determinism on traversal time. In traditional NoCs, a flit is routed hop by hop. Instead, in SHARP, a flit can be forwarded to the next stopping router with multiple (e.g., HPC_{max}) hops away in each cycle through the established bypass path. As stated in previous sections, our proposed scheduling policy supports priority-preemptive scheduling for hard real-time traffics in SHARP. Therefore, the stopping behavior at an intermediate router during a packet's transmission will take place resulting from the following two cases, shown in Fig. 2.

Case-1: Directly contending flow. In Fig. 2(a), the analyzed flow f_i shares resources (e.g., router, link) with a directly contending flow f_j from router R_3 to R_5; that is, the shared link set $sl_{ij} = \{\ell_3, \ell_4\} \neq \emptyset$. By considering the relative priority of such two flows, two cases are taken to analyze. In the first case, where f_j is a higher-priority flow, the analyzed packet of f_i will be preempted or lose arbitration when contending with a packet from f_j at R_3. In the second case, where f_j is the same-priority flow, the analyzed packet from f_i will be blocked at R_3 due to the first-in-first-out mechanism utilized when a f_j's packet is being transmitted over the shared resources. For the aforementioned two cases, the analyzed packet from f_i will be stopped at their first shared router R_3 because of suffered blocking. That is, the analyzed packet is always stopped at their first shared router where it contends with same/higher-priority flows in the worst scenario.

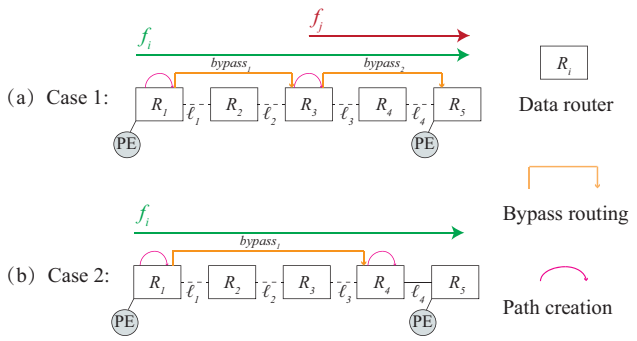


Fig. 2. Two stopping cases at intermediate routers for the analyzed flows f_i and f_j in priority-preemptive SHARP, HPC_{max} = 3.

Case-2: HPC_{max} limitation. Due to the transmission distance limitation of the electric signal, here we assume HPC_{max} = 3, a flit can only be forwarded with 3 physical hops within a cycle over the established data path. Therefore, as depicted in Fig. 2(b), f_i will be stopped at the HPC_{max}-limited stop router R_4, and recreate the remaining data path.

As discussed above, we can observe that the analyzed flow encounters stop at their first shared router in the worst case, due to the directly contending same/higher-priority flows and HPC_{max} limitation. We name the routers where the analyzed packet is buffered in the worst case as *stopping routers*, e.g., R_3, in Fig. 2(a). And other routers where the analyzed packets are bypassed are named *bypass routers*. That is to say, the bypass data path between two successive stopping routers seems to be directly connected by the point-to-point link, provided that the data path is established. In light of this observation, to reduce non-determinism and facilitate our real-time communication analysis, we can transform the analysis from multi-hop traversal into logically hop-by-hop traversal.

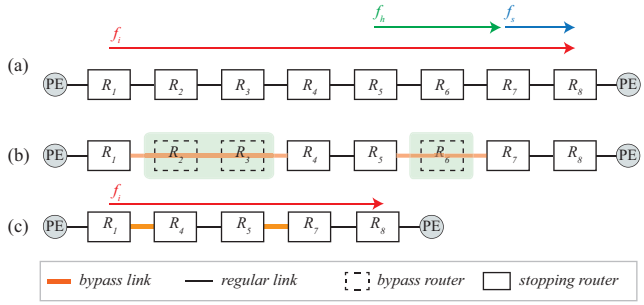


Fig. 3. An example of, for the analyzed flow f_i, flow graph transformation from multi-hop into logically hop-by-hop traversal, where the packet flows are in numbered priority order, $P_i = P_s > P_h$, and HPC_{max} = 3.

We continue to give an example of a flow set consisting of 3 flows transmitting in SHARP to demonstrate this flow graph transformation. A given set of flows including f_i, f_s and f_h are originally mapped in Fig. 3(a). By analyzing the transmitting behavior of the analyzed flow f_i in the worst communication case, the *bypass routers* are printed in a dotted rectangle, while *stopping routers* are printed in a solid rectangle, shown in Fig. 3(b). More specifically, for the analyzed flow f_i, it is stopped at R_4 due to HPC_{max} limitation, and encounters stops at R_5 and R_7 because of contending same (e.g., f_s) /higher (e.g., f_h) -priority flows. On top of it, we directly connect two successive

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3015440, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

stopping routers by a point-to-point link (namely bypass link). Then, based on that, we can derive the *new* flow graph of f_i with the feature of hop-by-hop traversal in Fig. 3(c). The real-time communication analysis in SHARP can be transformed from multi-hop into the "hop-by-hop" traversal. Note that, this transformation only facilitates real-time analysis with real-time scheduling theory at design time, which is a logical operation and will have no influence on actual performance at runtime.

For the analyzed flow f_i in priority-preemptive SHARP, we denote the transformed route as $path'_i$, and the worst-case basic communication latency C'_i along $path'_i$ can be formulated as:

$$C'_i = (t_r + t_w) \cdot |path'_i| + t_w \cdot (L_i - 1). \tag{3}$$

V. Real-Time Communication Analysis

Real-time analysis in traditional priority-preemptive wormhole NoCs has been conducted in several previous works [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]. The typical assumption in these works is the exclusive assignment of virtual channel (or priority level) to flows, and deadline restriction that a deadline is not greater than its period. However, the number of VCs is limited in practice, and in most cases, the number of flows is more than the number of VCs. The deadline may also be arbitrary (that is the deadline can be greater than its period). Especially, the state-of-the-art works [11], [12] didn't take these practical factors into account. To improve the state-of-the-art works [11], [12], we consider the shared VCs (or priority level) where multiple flows share the same priority or a group of VCs, similar to the literature [28], [29], [30], [31] in some traditional NoCs, and arbitrary-deadline flows based on the emerging multi-hop bypass traversal NoC. On such new priority-preemptive SHARP NoC, in this section, we conduct an elaborate and complete real-time analysis and derive the corresponding latency upper bound. This extension to SHARP is nontrivial due to more complex communication scenarios compared with traditional counterpart. We first give some basic concepts for the SHARP considering a shared priority. Next, we present the real-time communication analysis for constrained-deadline flows where their deadline is not greater than their period. Finally, we extend our real-time analysis to the arbitrary-deadline flows where their deadline may be greater than their period. Our analysis model both considers shared VCs (or priority levels) and arbitrary-deadline flows, which is more general and practical than state-of-the-art methods [10], [11], [12] in traditional NoCs and makes SHARP latency-predictable for real-time traffics.

A. Definitions

Unlike distinct priority policy, the shared priority policy permits multiple flows to share the same priority/a group of VCs, where each VC is specified a distinct priority. For the flows belonging to the same group of VCs, the first-in-first-out (FIFO) mechanism is utilized. Hence, for an analyzed flow f_i, we let $S(i)$ represent a set of flows that share the priority with f_i. Similar to the scenarios in traditional counterpart, direct interference, and indirect interference also exist in priority-preemptive SHARP and will be introduced later. Specifically,

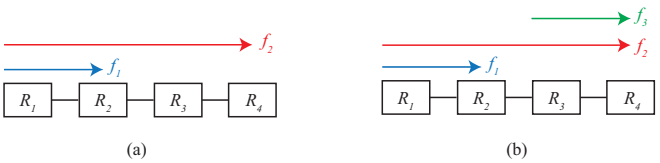


Fig. 4. An example to illustrate direct and indirect interference/blocking.

there are four different types of blocking/interference that should also be considered in our analysis. Before discussing the real-time analysis model, we first give some definitions. In order to differentiate the blocking/interference latency from higher-priority or same-priority flows, we let the *blocking* denote the blocking latency suffered from the contending *same-priority flows*, and *interference* denote the interference latency suffered from the contending *higher-priority flows*.

Definition 1 (Direct Interference [1], [2])**.** *If the analyzed flow f_i has a lower priority than that of flow f_j, and shares at least one link with f_j, then f_i suffers the* direct interference *from f_j. The direct flow set of f_i is defined as $S_D(f_i)$:*

$$S_D(f_i) = \{f_j | sl_{ij} \neq \emptyset, P_i > P_j, \forall f_j \in \Gamma\}$$

Example 1. *In the example of Fig. 4(a), assuming f_1 has a higher priority than f_2, the analyzed flow f_2 will suffer direct interference from the contending higher-priority flow f_1 due to spatial contention (e.g., link $R_1 \rightarrow R_2$).*

Definition 2 (Direct Blocking)**.** *Inspired by the direct interference, if the analyzed flow f_i has the same priority with flow f_j, and shares at least one link with f_j under the mechanism of FIFO, then f_i suffers the direct blocking from f_j. The direct blocking flow set of f_i is defined as $S_{SD}(f_i)$:*

$$S_{SD}(f_i) = \{f_j | sl_{ij} \neq \emptyset, P_i = P_j, \forall f_j \in \Gamma\}$$

Example 2. *In the example of Fig. 4(a), assuming f_1 has the same priority as f_2, the analyzed flow f_2 will suffer direct blocking from the contending same-priority flow f_1 in the worst case.*

Definition 3 (Indirect Interference [2])**.** *If the analyzed flow f_i has no shared links with a higher-priority flow f_k, but suffers direct interference from an intermediate flow f_j that suffers direct blocking/interference from f_k, then f_i suffers the indirect interference from f_k via the intermediate flow f_j. The indirect flow set of f_i is defined as $S_I(f_i)$:*

$$S_I(f_i) = \{f_k | sl_{ik} = \emptyset, \forall f_j \in S_D(f_i), \forall f_k \in (S_D(f_j) \cup S_{SD}(f_j))\}$$

Example 3. *In the example of Fig. 4(b), assuming the priority order is: $p_1 \leq p_2 < p_3$ (i.e., the smaller value represents a higher priority), f_3 will suffer indirect interference from the higher-priority flow f_1 via the intermediate flow f_2.*

Definition 4 (Indirect Blocking)**.** *Inspired by the indirect interference, assume the analyzed flow f_i has no shared links with f_k that has a same or higher priority, but suffers direct blocking from an intermediate flow $f_j \in S_{SD}(f_i)$ that suffers direct blocking/interference from f_k, then f_i suffers the*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3015440, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

indirect blocking from f_k through an intermediate flow f_j. The indirect blocking flow set is defined as $S_{SI}(f_i)$:

$$S_{SI}(f_i) = \{f_k | sl_{ik} = \emptyset, \forall f_j \in S_{SD}(f_i), \forall f_k \in (S_D(f_j) \cup S_{SD}(f_j))\}$$

Example 4. *In the example of Fig. 4(b), assuming the priority order is: $p_1 \leq p_2 = p_3$, f_3 will suffer indirect blocking from the same/higher-priority flow f_1 via the intermediate flow f_2.*

Note that, in priority-preemptive SHARP, the indirect interference suffered by f_3 influences the number of packets of f_2 that causes direct interference to f_3. While the indirect blocking influences the actual release time of f_2 that causes blocking to f_3. These two kinds of indirect blocking/interference will be formally considered later.

Definition 5 (Downstream Indirect Interference [6], [10])**.** *The flow $f_j \in S_D(f_i)$ may cause direct interference to the analyzed flow f_i more than once at the downstream position, thus leading to the suffered interference each time being more than zero-load latency C_j. The downstream indirect interference flow set is defined as $S_I^{dn_j}(f_i)$:*

$$S_I^{dn_j}(f_i) = \{f_k | f_k \in S_I(f_i) \cap (S_D(f_j) \cup S_{SD}(f_j)), last(sl_{ij}, path_j) < first(sl_{jk}, path_j)\}$$

*where the shared links between f_k and f_j appears **after** the shared links between f_j and f_i; the function $first(sl_{ij}, path_j)$ (resp. $last(sl_{ij}, path_j)$) returns the subscript of the first (resp. last) shared link along the route $path_j$ of flow f_j.*

Example 5. *In the example of Fig. 5, assuming the priority order is: $p_1 \leq p_2 < p_3$, f_3 will be interfered by a packet of the contending higher-priority flow f_2 more than once (e.g., R_2) in the worst case because of f_1 and limited router buffer size.*

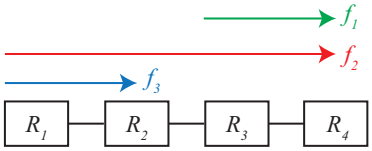


Fig. 5. An example to illustrate downstream indirect interference/blocking.

Definition 6 (Downstream Indirect Blocking)**.** *Inspired by the downstream indirect interference, f_i can be blocked by a direct blocking flow f_j more than once under the FIFO mechanism. That is, the flits of f_j (that has already caused blocking to f_i) are buffered at their shared routers due to suffered blocking/interference from an indirect blocking flow $f_k \in S_{SI}(f_i)$, then such buffered flits of f_j may block f_i again when the router buffer size is limited. The downstream indirect blocking flow set $S_{SI}^{dn_j}(f_i)$ is defined as:*

$$S_{SI}^{dn_j}(f_i) = \{f_k | f_k \in S_{SI}(f_i) \cap (S_D(f_j) \cup S_{SD}(f_j)), last(sl_{ij}, path_j) < first(sl_{jk}, path_j)\}$$

where the function $first(sl_{ij}, path_j)$ (resp. $last(sl_{ij}, path_j)$) return the subscript of the first (resp. last) shared link along the route $path_j$ of flow f_j.

Example 6. *In the example of Fig. 5, assuming the priority order is: $p_1 \leq p_2 = p_3$, f_3 will be blocked by a packet of the contending higher-priority flow f_2 more than once in the worst case because of f_1 and limited router buffer size.*

Note that, the downstream indirect blocking/interference suffered by f_3 influence the maximum blocking/interference of each packet from f_2 that causes to f_3.

B. Real-Time Analysis for Constrained-Deadline Flows

In this section, we consider the real-time communication analysis for constrained-deadline flows where their deadlines are not greater than their periods ($D_i \leq T_i$). As stated in Section IV-B, the on-chip routers of SHARP are classified as *stopping router* where the analyzed packet is buffered in the worst communication scenario, and *bypass router* where the analyzed packet can bypass. Through logical communication flow graph transformation, the original route $path_i$ of the analyzed flow f_i is transformed into $path_i'$. Due to priority-preemptive communication scheduling utilized in SHARP, except the new zero-load latency C_i', an analyzed flow f_i suffers queue blocking SB_i from same-priority flows, and interference HI_i from higher-priority flows. Therefore, the worst-case end-to-end latency (WCEL) R_i is formulated as:

$$R_i = C_i' + SB_i + HI_i \tag{4}$$

Lemma 1 (Ref [32])**.** *For the constrained-deadline packet flows, an analyzed packet $f_{i,\lambda}$ can be blocked by a same-priority flow $f_s (f_s \in S_{SD}(f_i))$ at most one packet, even though the indirect blocking exists.*

Theorem 1. *When calculating WCEL, for the analyzed flow packet $f_{i,\lambda}$, the upper bound of SB_i suffered from the same-priority flows $f_s (f_s \in S_{SD}(f_i))$ is given by:*

$$SB_i = \begin{cases} \sum_{f_s \in S_{SD}(f_i)} C_s', & S_{SI}^{dn_s}(f_i) = \phi, \\ \sum_{f_s \in S_{SD}(f_i)} (C_s' + B_{si}^{dn}), & otherwise. \end{cases} \tag{5}$$

where the additional blocking $B_{si}^{dn} = min(dib_{is}, R_s - C_s')$, the downstream indirect blocking (defined in Definition 6) $dib_{is} = buff \cdot t_w \cdot |sl_{is}|$ and the parameter buff refers to the buffer depth for each virtual channel of the on-chip router.

Proof. When considering the relative position of the shared links between the contending same-priority flows f_s and the analyzed flow f_i, the indirect blocking is divided into upstream and downstream indirect blocking, shown in Fig. 6.

Fig. 6. (a) Upstream and (b) downstream indirect blocking.

Case-a: In Fig. 6(a), f_i suffers upstream indirect blocking from $f_u (f_u \in S_{SD}(f_s) \cup S_D(f_s))$ via an intermediate same-priority flow f_s. In the worst case, the generated packets from f_s (e.g., $f_{s,\beta}$) suffer enough blocking/interference from f_u and

arrive at their first shared router (e.g., R_3) between f_i and f_s just before the release time of $f_{i,\lambda}$. Then, $f_{i,\lambda}$ is blocked by $f_{s,\beta}$ at their first shared router until the shared resources are available. And by Lemma 1, f_s actually blocks $f_{i,\lambda}$ at most one packet (e.g., $f_{s,\beta}$) in this case. That is, the upstream indirect blocking only affects the actual release time of the intermediate flow f_s, not the number of packets of f_s that causes blocking to $f_{i,\lambda}$. Hence, the suffered blocking from f_s is upper-bounded by C'_s for the upstream case.

Case-b: In Fig. 6(b), f_i suffers downstream indirect blocking from $f_d(f_d \in S_{SI}^{dn_s}(f_i))$ via an intermediate same-priority flow f_s. Similar to downstream indirect interference [10], [11], an intermediate packet $f_{s,1}$ that has caused blocking at their first shared router (e.g., R_1) can block $f_{i,\lambda}$ again at later shared routers (e.g., R_2) due to indirect blocking. This is because, the buffer size of on-chip routers is finite in practice and $f_{s,1}$ may be blocked by f_d at R_3, some flits of $f_{s,1}$ may be buffered along the shared intermediate routers between f_s and f_i. Such buffered flits can cause blocking to $f_{i,\lambda}$ again due to the utilized first-in-first-out mechanism in the shared virtual channels. We call such additional blocking as *downstream indirect blocking* denoted by dib_{is}. Suppose the packet of f_s is fully buffered all of their shared stopping routers between f_s and f_i, the maximum value of dib_{is} suffered by $f_{i,\lambda}$ is $dib_{is} = buff \cdot t_w \cdot |sl_{is}|$. On the other hand, $f_{s,1}$ suffers direct blocking/interference from f_d up to $R_s - C'_s$, which in turn blocks $f_{i,\lambda}$. Since such indirect blocking may be not enough to make all the buffers of intermediate shared routers between f_s and f_i filled up, the maximum additional downstream indirect blocking suffered by $f_{i,\lambda}$ is not larger than $min(dib_{is}, R_s - C'_s)$. Therefore, in this case, the suffered queue blocking from $f_{s,1}$ is upper-bounded by $C'_s + min(dib_{is}, R_s - C'_s)$ for the downstream case.

Based on the above two cases, the theorem follows. □

Similar to the real-time analysis in traditional counterpart, the flows in priority-preemptive SHARP also suffer direct and indirect interferences. The interference HI_i suffered by the analyzed flow f_i is the summation of HI_{hi} caused by direct higher-priority flows $f_h(f_h \in S_D(f_i))$. The value of HI_{hi} is given by the result of multiplication between the number of packets from f_h that causes interference to f_i and the maximum interference that a packet from f_h causes to f_i.

Theorem 2. *When calculating WCEL, for the analyzed packet $f_{i,\lambda}$, the upper bound of HI_{hi} suffered from the higher-priority flow $f_h(f_h \in S_D(f_i))$ is given by:*

$$HI_{hi} = \left\lceil \frac{R_i + J_h + I_{hi}^I}{T_h} \right\rceil \cdot (C'_h + I_{hi}^{dn}). \tag{6}$$

where

$I_{hi}^I = R_h - C'_h$,

$I_{hi}^{dn} = \sum_{f_k} \left\lceil \frac{R_h + J_k}{T_k} \right\rceil \cdot min(dii_{ih}, C'_k + I_{kh}^{dn}) + SB_h^q$.

($f_k \in S_I^{dn_h}(f_i), f_k \notin S_{SI}^{dn_s}(f_i), f_s \in S_{SD}(f_i)$, $f_q \in S_{SD}(f_h)$, f_q shares links with f_h at the downstream position of f_h), and

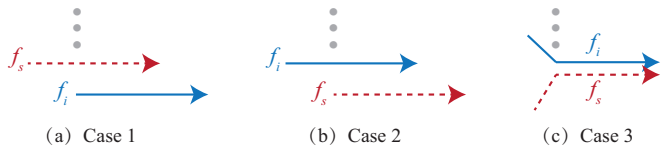
$dii_{ih} = buff \cdot t_w \cdot |sl_{ih}|$.

Proof. For $f_{i,\lambda}$, let direct higher-priority flow be $f_h \in S_D(f_i)$. Since SHARP enables priority-preemptive scheduling, similar to the analysis in traditional NoCs [10], [11], multiple packets from f_h can cause interference to $f_{i,\lambda}$ and the downstream indirect interference [6], [10] also exists during $f_{i,\lambda}$'s transmission. The maximum number of packets from f_h is upper-bounded by $\left\lceil \frac{R_i + J_h + I_{hi}^I}{T_h} \right\rceil$ that is formally proved in [10], and the additional downstream interference I_{hi}^{dn} from a packet of f_h by I_{hi}^{dn} that is partly calculated in [11]. Besides, f_h additionally suffers blocking from downstream same-priority flows f_q that in turn causes additional interference to $f_{i,\lambda}$, which is calculated by SB_h^q. However, since the flows $f_k \in S_I^{dn_h}(f_i) \cap S_{SI}^{dn_s}(f_i)$ has caused blocking via intermediate same-priority flow $f_s \in S_{SD}(f_i)$, which have been considered in Theorem 2, this flow $f_k \in S_I^{dn_h}(f_i) \cap S_{SI}^{dn_s}(f_i)$ is not included here. By adding the new zero-load latency C'_h, the maximum suffered interference of each encounter is $C'_h + I_{hi}^{dn}$. By accumulating all calculated parameters, the theorem follows. □

Substituting Eq. (5) and (6) into Eq. (4), the latency upper bound R_i is derived for constrained-deadline flows.

C. Real-Time Analysis for Arbitrary-Deadline Flows

Instead of the packet-level analysis for constrained-deadline flows in Section V-B, we will consider the real-time analysis at the flow-level for arbitrary-deadline flows (the deadline restriction $D_i \leq T_i$ is relaxed, namely D_i can be $(<, =, >)$ T_i) in this section. In this case, for the packets of the same analyzed flow, the transmission of a previous successive packet may not be completed when the next analyzed packet is released. Since the first-in-first-out mechanism is utilized for the flows with the same priority, the earlier packet may delay the later ones from the same flow. To calculate *WCEL* of an arbitrary-deadline flow f_i, we extend the *busy period* [33], [21] utilized for task scheduling in uniprocessor systems to flow scheduling in priority-preemptive SHARP. During the *busy period* of the analyzed flow f_i, all of the packets of f_i and its contending flows of priority P_i or higher finish transmission. To derive *WCEL*, for the analyzed flow f_i, we firstly find the upper bound of the busy period $R_i(\lambda)$ that is from the critical instant and ends until the completion of the λ^{th} packet $f_{i,\lambda}$. Then the final *WCEL* of the analyzed flow is chosen as the maximum *WCEL* among the λ packets.



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 7. The relative position relationships between the analyzed flow f_i and its same-priority flow f_s.

Lemma 2. *For the arbitrary-deadline packet flows, an analyzed packet $f_{i,\lambda}$ can be blocked by a same-priority flow $f_s(f_s \in S_{SD}(f_i))$ more than one packet.*

Proof. For the two contending flows f_i and f_s, there are three cases of the relative position relationship, shown in Fig. 7.

For any case of Fig. 7, suppose $f_{i,\lambda}$ is released at time t_0, while $f_{s,1}$ is actually released at their first shared router at $t_0 - \epsilon$ ($\epsilon > 0$) after experienced direct blocking/interference (correspondingly the indirect blocking for f_i). $f_{i,\lambda}$ will be blocked by $f_{s,1}$ until $f_{s,1}$ departs from that shared links sl_{is} at time $t_0 - \epsilon + C_s$ (where C_s is the duration time that $f_{s,1}$ departs from the shared links). Suppose the κ^{th} packet $f_{s,\kappa}$ of f_s is released at t_1. Since D_i $(<,=,>)$ T_i, the value of $t_0 - \epsilon + C_s$ can be larger than t_1 (i.e., a later period start time of f_s).

Hence, there may be multiple packets (e.g., κ) ahead of $f_{i,\lambda}$. Since the FIFO mechanism is employed and preemption is not allowed between any two flows with the same priority, the analyzed packet $f_{i,\lambda}$ is blocked by multiple packets until the shared resources are available. The lemma is proved. □

Theorem 3. *The upper bound $R_i(\lambda)$ of a busy period for the analyzed flow f_i with an arbitrary deadline, under the shared priority policy in priority-preemptive SHARP, is given by:*

$$R_i(\lambda) = \lambda C_i' + \sum_{\forall f_s \in S_{SD}(f_i)} \left\lceil \frac{R_i(\lambda) + J_s}{T_s} \right\rceil \cdot SB_i + \sum_{\forall f_h \in S_D(f_i)} \left\lceil \frac{R_i(\lambda) + J_h + I_{hi}^I}{T_h} \right\rceil \cdot (C_h' + I_{hi}^{dn}). \tag{7}$$

where $\lambda = \left\lceil \frac{R_i(\lambda)+J_i}{T_i} \right\rceil$.

Proof. The busy period $R_i(\lambda)$ is the time interval from the release time of $f_{i,1}$ until the transmission completion of $f_{i,\lambda}$, which consists of four parts: total basic latency from the analyzed flow f_i's λ packets, and blocking/interference from directly contending same-priority flows f_s and higher-priority flows f_h. During the busy period, the number of packets from the analyzed flow f_i is calculated by $\lambda = \left\lceil \frac{R_i(\lambda)+J_i}{T_i} \right\rceil$. In the following, we use mathematical induction to prove the Equation (7) for $\forall j \in \{1, 2, ..., \lambda\}$.

Base Case: For the first packet from $f_{i,1}$, let the release time of $f_{i,1}$ as the critical instant of $R_i(1)$.

First, for the queue blocking from the same-priority flows, according to Lemma 2, $f_{i,1}$ can be blocked by $\left\lceil \frac{R_i(1)+J_s}{T_s} \right\rceil$ packets from f_s during the relative transmission interval $R_i(1)$ of $f_{i,1}$. While $f_{i,1}$ can suffer queue blocking from each of these packets up to SB_i that is proved in Theorem 2.

Second, for the interference from higher-priority flows, according to Theorem 3, such interference from a higher-priority flow is upper-bounded by HI_{hi}.

Therefore, Equation (7) is satisfied when $j = 1$.

Inductive Step: For the packet $f_{i,k}$, assume that Equation (7) is satisfied when $\forall k \in \{1, 2, ..., j-1\}$. We need to prove Equation (7) is satisfied when $k = j$.

First, for the basic latency of *k* packets during the relative time interval $R_i(k)$, such latency is upper-bounded by $k \cdot C_i'$. Based on the previous *j-1* packets (when $k = j-1$), the basic latency of *j* packets is upper-bounded by $(j-1) \cdot C_i' + C_i' = j \cdot C_i'$.

Second, for the queue blocking from the same-priority flows, the number of packets from f_s is upper-bounded by $\left\lceil \frac{R_i(j)+J_s}{T_s} \right\rceil$ during the relative time interval $R_i(j)$, and the blocking of each packet is upper-bounded by SB_i according to Theorem 2.

Third, for the interference from higher-priority flows, the number of packets from f_h is upper-bounded by $\left\lceil \frac{R_i(j)+J_h+I_{hi}^I}{T_h} \right\rceil$, and the maximum interference of each encounter is $C_h' + I_{hi}^{dn}$.

In summary, Equation (7) is satisfied when $k = j$. When $j = \lambda$, the theorem is proved. □

By using the busy period upper bound $R_i(\lambda)$, the worst-case end-to-end latency (WCEL) R_i of the analyzed flow f_i is given by the packet with maximum analytical latency upper bound. That is, R_i is given by:

$$R_i = \max_{\forall n \in [1,\lambda]} (R_i(n) - (n-1)T_i) + J_i. \tag{8}$$

In summary, regarding the calculation of the analytical latency upper bound R_i of f_i, which method to be adopted depends on the employed task model. For the constrained-deadline flows, R_i is calculated by Equation (4); while for the arbitrary-deadline flows, Equation (8) is adopted.

D. Qualitative Advantages of Our Approach

In this section, we analytically compare the priority-preemptive SHARP and the schedulability analysis with that of traditional hop-by-hop traversal wormhole NoCs [11]. Note that we declare that we cannot compare with the work [12], since we don't consider these aspects temporarily (e.g., a finer-grained sub-route interference [7]) to concentrate on our research target of this paper, and our methods can be easily extended to consider the finer-grained sub-route interference [7]. Since the work [11], [12] only considers the constrained-deadline flows, we here also compare the analytical bounds for constrained-deadline flows. For the analyzed constrained-deadline flow f_i, the worst-case end-to-end latency R_i^{sharp} in priority-preemptive SHARP is formulated as follows:

$$R_i^{sharp} = C_i' + \sum_{f_s \in S_{SD}(f_i)} (C_s' + B_{si}^{dn,sharp}) + \sum_{f_h \in S_D(f_i)} \left(\left\lceil \frac{R_i^{sharp} + J_h + I_{hi}^{I,sharp}}{T_h} \right\rceil \cdot (C_h' + I_{hi}^{dn,sharp}) \right). \tag{9}$$

where

$B_{si}^{dn,sharp} = min(dib_{is}^{sharp}, R_s^{sharp} - C_s')$,

$dib_{is}^{sharp} = buff \cdot t_w \cdot |sl_{is}|$,

$I_{hi}^{I,sharp} = R_h^{sharp} - C_h'$,

$I_{hi}^{dn,sharp} = \sum_{f_k} \left\lceil \frac{R_h+J_k}{T_k} \right\rceil \cdot min(dii_{ih}^{sharp}, C_k' + I_{kh}^{dn}) + SB_h^{q,sharp}$ ($f_k \in S_I^{dn,sharp_h}(f_i), f_k \notin S_{SI}^{dn,sharp_s}(f_i), f_s \in S_{SD}(f_i)$, $f_q \in S_{SD}(f_h)$, f_q shares links with f_h at the downstream position of f_h),

$dii_{ih}^{sharp} = buff \cdot t_w \cdot |sl_{ih}|$.

While for traditional hop-by-hop traversal wormhole NoCs [11], we also employ the priority shared policy based on that and derive the worst-case end-to-end latency R_i^{tradi}:

$$R_i^{tradi} = C_i + \sum_{f_s \in S_{SD}(f_i)} (C_s + B_{si}^{dn,tradi}) + \sum_{f_h \in S_D(f_i)} \left(\left\lceil \frac{R_i^{tradi} + J_h + I_{hi}^{I,tradi}}{T_h} \right\rceil \cdot (C_h + I_{hi}^{dn,tradi}) \right). \quad (10)$$

where

$B_{si}^{dn,tradi} = min(dib_{is}^{tradi}, R_s^{tradi} - C_s)$,

$dib_{is}^{tradi} = buff \cdot t_w \cdot |sl_{is}|$,

$I_{hi}^{I,tradi} = R_h^{tradi} - C_h$,

$I_{hi}^{dn,tradi} = \sum_{f_k} \left\lceil \frac{R_h + J_k}{T_k} \right\rceil \cdot min(dii_{ih}^{tradi}, C_k + I_{kh}^{dn,tradi}) + SB_h^{q,tradi} \ (f_k \in S_I^{dn,tradi_h}(f_i), f_k \notin S_{SI}^{dn,tradi_s}(f_i), f_s \in S_{SD}(f_i))$,

$dii_{ih}^{tradi} = buff \cdot t_w \cdot |sl_{ih}|$.

By analyzing the analytical bound R_i, it essentially depends on $path_i$, router-stage latency t_r, wire (between two successive stopping routers) latency t_w, buffer depth *buff*, release jitter J_i, and contending same/higher-priority flows. In SHARP with network frequency 1-2GHz at 45nm [13], [16], the worst-case router-stage latency $t_r^{sharp} = 2$ cycles, while the best-case router-stage latency is 1 cycle when no buffering flits exist at start/source router. In traditional NoCs [34], [35], [36], the router-stage latency t_r^{tradi} ranges from 1 to 5 cycles because of different techniques (i.e., network frequency). Besides the variables $path_i$ and t_r, all the other variables are the same in traditional and SHARP NoCs. In Section IV-B, the route $path_i'$ is transformed from $path_i$ with $|path_i'| \leq |path_i|$. The latency bounds are directly proportional to the basic zero-load latency C_i, and thus we mainly compare the variable C_i.

Since

$$C_i = (t_r + t_w) \cdot |path_i| + t_w \cdot (L_i - 1).$$

By analytically comparing the $path_i$ and t_r in traditional and SHARP NoCs, we can conclude:

Theorem 4. *R_i^{sharp} strictly dominates R_i^{tradi} when $t_r^{tradi} \geq 2$; while R_i^{sharp} partly dominates R_i^{tradi} when $t_r^{tradi} = 1$.*

Proof. Case-1: $t_r^{tradi} \geq 2$. We prove this case in two steps: (1.1) prove R_i^{sharp} dominates R_i^{tradi}, i.e., R_i^{sharp} is no larger than R_i^{tradi}, and (1.2) further prove R_i^{sharp} strictly dominates R_i^{tradi}, i.e., R_i^{sharp} is strictly less than R_i^{tradi}. For (1.1), since $|path_i'| \leq |path_i|$ and $t_r^{sharp} \leq t_r^{tradi}$, we get $C_i^{sharp} \leq C_i^{tradi}$. Since other variables are the same, we then derive $R_i^{sharp} \leq R_i^{tradi}$. Step (1.1) is proved. As for (1.2), for an analyzed flow f_i with $|path_i| > 1$, by considering an extreme case where the analyzed flow bypasses all the intermediate routers, we derive $|path_i'| < |path_i|$, $C_i^{sharp} < C_i^{tradi}$. In this case, $R_i^{sharp} < R_i^{tradi}$. Step (1.2) is proved.

Therefore, when $t_r^{tradi} \geq 2$, R_i^{sharp} strictly dominates R_i^{tradi}.

Case-2: $t_r^{tradi} = 1$. In this case, traditional NoCs have a larger variable $|path_i|$, while SHARP NoCs have a larger router-stage latency t_r. Consider two extreme cases. (2.1) when an analyzed flow f_i with $|path_i| = 1$ and $t_r^{sharp} = 2$, $|path_i|' = 1$. SHARP NoCs have no benefits from $|path_i|'$ reduction, and thus R_i^{tradi} strictly dominates R_i^{sharp}. (2.2) when an analyzed flow f_i with $|path_i| > 1$, and f_i bypasses all the intermediate routers namely $|path_i|' = 1$, we derive $C_i^{tradi} - C_i^{sharp} = 2 \cdot |path_i| - 3 \cdot |path_i|' > 0$. $C_i^{tradi} > C_i^{sharp}$. In this case, $R_i^{sharp} < R_i^{tradi}$. Therefore, when $t_r^{tradi} = 1$, R_i^{sharp} partly dominates R_i^{tradi}.

□

Note that, the state-of-the-art works [11], [12] didn't consider the priority/VCs shared policy and the general flows with an arbitrary deadline. In summary, based on Theorem 4, the advantages of our approach in SHARP are threefold.

- The communication performance of SHARP is better than that of traditional NoCs since SHARP can enable single-cycle multi-hop traversal. Specifically, SHARP NoCs strictly dominate traditional NoCs with deep pipeline router (i.e., $t_r^{tradi} \geq 2$); while for the state-of-the-art academic traditional NoCs with 1-cycle router (i.e., $t_r^{tradi} = 1$), SHARP NoCs dominate if the benefits from $|path_i|'$ reduction are more than the penalty of router-stage latency increasing. In particular, when the real-time flows have very tight deadlines, SHARP can provide a lower worst-case end-to-end latency and then guarantee better schedulability than traditional NoCs.
- Our analysis approach has considered the priority/VCs shared policy, which is more practical than the work [11].
- Our analysis approach extends the real-time analysis to arbitrary-deadline flows, which is more general and can also apply to traditional NoCs.

As mentioned in Section IV-A, for an analyzed flow f_i, if the analytical latency upper bound R_i meets the condition $R_i \leq D_i$, any generated packet from f_i can be *schedulable* at runtime under the worst communication case. Since the original bound R_i is reduced to R_i' with the help of our proposal ($R_i' \leq R_i$), the flow sets that are originally *not schedulable* (formally $R_i > D_i$) can be potentially *schedulable* (formally $R_i' \leq D_i$), thus improving the real-time schedulability.

VI. Quantitative Evaluation

To quantitatively evaluate the real-time performance of the priority-preemptive SHARP, we conduct experimental comparisons by using our proposed analytical model under synthetic benchmarks. We first conduct a set of experiments to contrast the normalized latency bounds of the priority-preemptive SHARP against the results of original distance-based SHARP and two kinds of traditional hop-by-hop traversal NoCs with different router-stage latency (i.e., $t_r^{tradi} = 1, 2$). Then, we conduct another set of experiments to verify the schedulability improvement, compared with the aforementioned baselines.

A. Experimental Setup

To validate our proposal, we conduct the quantitative evaluations by using a large set of synthetic packet flows [25] (e.g.,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.3015440, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

UniformRandom traffic pattern) and compute their latency upper bounds of our proposed and existing analytical models. The configurations of the communication platforms and real-time packet flows are set as follows:

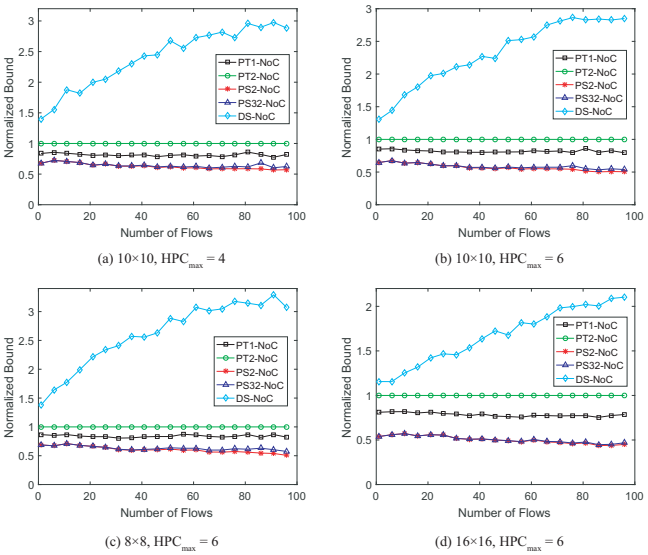
1) We use $N \times N$ mesh size 2D-NoCs (i.e., $N = 8, 10, 16$) with a design-time parameter HPC_{max} (i.e., HPC_{max} = 4, 6), for SHARP NoCs, which denotes the maximum number of bypass hops per clock cycle.
2) XY-routing is used. Wormhole flow control is adopted for priority-preemptive traditional and SHARP NoCs, while virtual cut-through flow control is adopted for original distance-based SHARP NoCs.
3) The buffer depth of each VC in priority-preemptive traditional and SHARP NoCs is maximally limited to 32 flits; while for original distance-based SHARP, the buffer depth is equal to the largest packet size.
4) The number of packet flows is in the range of (1, 100) with the step of 5. Each packet size in flits is randomly selected from the range [5, 50].
5) The utilization of each packet flow is selected from the range (0.01, 0.5). The corresponding period T_i is set as the ratio of the zero-load latency to its utilization. The deadline D_i is set as the same as the period (i.e., $D_i = T_i$). The release jitter J_i is set as 0.
6) In the state-of-the-art work [11], the constrained-deadline flows (i.e., $D_i \leq T_i$) only are considered. It assumes the priority of each flow is unique, and the number of VCs is more than the number of priority levels. To facilitate comparison, we also only consider the constrained-deadline flows and assume the number of priority levels are equal to the maximum number of flows, while 6 VCs per priority level is achieved to cover the buffer turnaround time. The priorities of the flows are assigned by Rate Monotonic (RM) scheduling [26], where a flow with a shorter period has higher priority.
7) Each point setting of the result figures is conducted 100 times, and the average value is used as the final value.

The proposed approach, priority-preemptive scheduling policy and its corresponding real-time analytical model in SHARP NoCs ($t_r^{sharp} = 2$) with the buffer depth (2 and 32 flits), will be referred as PS2-NoC and PS32-NoC henceforth, respectively. While the considered baselines include:

- PT1-NoC. Priority-preemptive Traditional hop-by-hop traversal NoCs with router-stage latency $t_r^{tradi} = 1$.
- PT2-NoC. Priority-preemptive Traditional hop-by-hop traversal NoCs with router-stage latency $t_r^{tradi} = 2$.
- DS-NoC. Original distance-based SHARP NoCs.

To facilitate comparison, we use the following performance metrics for quantitative comparison in our experiments:

- *Normalized Bound.* To demonstrate the real-time performance in terms of bounds reduction of our method denoted by PS-NoC, we employ the *normalized bound* to compare with PT1-NoC, PT2-NoC, and DS-NoC. In our experiments, all the analytical worst-case end-to-end latency bounds are normalized to that of PT2-NoC.
- *Schedulable Flows(%).* For each analyzed flow set, we employ the metric *schedulable flows(%)* to show the achieved schedulability improvement compared with the baselines. The *schedulable flows(%)* metric is defined as the percentage of schedulable flows for an analyzed flow set under a certain configuration.



(a) 10×10, HPC_{max} = 4 (b) 10×10, HPC_{max} = 6

(c) 8×8, HPC_{max} = 6 (d) 16×16, HPC_{max} = 6

Fig. 8. Normalized latency upper bound comparisons with different configurations against the traditional counterparts and original distance-based SHARP using synthetic *UniformRandom* traffic pattern [25].

B. Evaluation Results

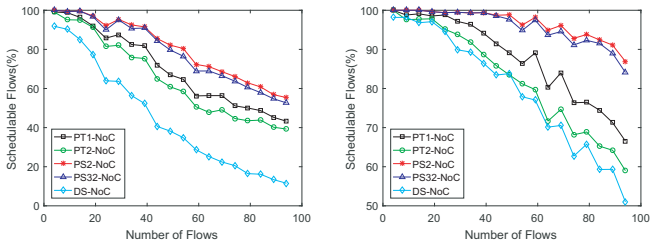
Fig. 8 demonstrates the normalized bounds between our proposed method denoted by PS-NoC and the compared existing methods under different topology and HPC_{max}. From the experimental results, compared with PT1-NoC, PT2-NoC, and DS-NoC, the normalized bounds of PS2-NoC are averagely reduced with 29.24%, 42.39% and 74.40%, while PS32-NoC achieves for 27.23%, 40.75% and 73.68%, respectively. The reason is that we employ the priority-preemptive scheduling policy based on single-cycle multi-hop traversal NoCs, thus reducing the suffered blocking latency from other contending flows for higher-priority flows and breaking the distance limitation between long-distance communication packet flows. More specifically, compared with the baselines through the shown results in Fig. 8, the observations are fourfold.

- Compared with PT2-NoC, as proved in Theorem 4, our proposal strictly dominates PT2-NoC. This is because, with the help of utilization of single-cycle multi-hop bypass path in SHARP, the zero-load latency is reduced significantly, and accordingly the suffered blocking latency from higher-priority flows is reduced.
- Regarding PT1-NoC, as proved in Theorem 4, whether our proposal dominates PT1-NoC or not depending on the relative size in SHARP between the benefit of hop count (i.e., $|path_i|$) reduction and penalty of router-stage latency (i.e., t_r) increasing. Note that, we indeed find that PT1-NoC dominates our proposal in a small number of experimental cases, while our proposal dominates PT1-NoC in the rest of cases. It means that the benefit of

hop count reduction is more than the penalty of router-stage latency increasing in most cases. Hence, the average bounds of our proposal are smaller than that of PT1-NoC.

- As for DS-NoC, our proposal significantly outperforms in the same SHARP but with different scheduling. The reason is that our proposal can provide differentiated service for real-time flows, while DS-NoC is generally optimized for average performance for best-effort flows. Therefore, the higher-priority flows of our proposal suffer less blocking latency when encountering contention than that of DS-NoC. In particular, DS-NoC shows even larger normalized bounds than PT2-NoC, indicating the enabled differentiated service is very crucial for real-time flows.
- The real-time performance of our proposal with different buffer depth is slightly different. Specifically, the normalized bounds of PS2-NoC (with a smaller 2-flit buffer depth) are averagely 2.76% lower than that of PS32-NoC (with a larger 32-flit buffer depth). This is because, the approach PS32-NoC with a larger buffer depth can bring more downstream indirect interference in the worst case. Thus, a larger buffer depth of each VC is not beneficial for real-time traffics.

Fig. 9 shows the schedulability results against the baselines under two mesh sizes. From the experimental results, compared with PT1-NoC, PT2-NoC, and DS-NoC, the percentage of schedulable flows of PS32-NoC is averagely improved by 10.52%, 18.39% and 40.61%, while PS2-NoC achieves for 12.12%, 20.11% and 42.65%, respectively. As mentioned in Fig. 8, this is due to the reduced latency, the originally unschedulable flows in the baselines are potentially able to become schedulable in our proposal (i.e., PS2-NoC, PS32-NoC), thus making a higher percentage of flows deadline guaranteed (namely schedulable with $R_i \leq D_i$). Compared with PS32-NoC, PS2-NoC achieves averagely 1.48% more number of schedulable flows due to lower suffered downstream indirect interference. Besides, with the increasing of the number of flows, both sub-figures of Fig. 9 demonstrate that the *schedulable flows* metric decreases due to more suffered blocking latency. In summary, by combining with the matched priority-preemptive scheduling based on the emerging multi-hop bypass NoCs, the schedulability is remarkably improved.



(a) 8×8, $HPC_{max} = 6$ (b) 16×16, $HPC_{max} = 6$

Fig. 9. Schedulability results for our proposal against the baselines.

VII. Conclusion

To reduce the worst-case end-to-end latency for real-time traffics, we employed the latest evolved SMART NoC, called SHARP, as the communication infrastructure in real-time multi/many-core systems, and proposed matched priority-preemptive scheduling. To facilitate analysis and validate schedulability, a corresponding real-time communication analysis model, applicable for constrained and arbitrary-deadline flows, is also proposed by considering priority/VCs share. Communication latency is reduced significantly by using SHARP, and worst-case latency predictability is guaranteed by using our analysis model. Through theoretical and experimental comparison, the latency bounds in our proposal dominate that of traditional hop-by-hop traversal NoCs and original distance-based SHARP, especially for deep pipeline routers. Combined with the proposed priority-preemptive scheduling policy and real-time analysis model, SHARP shows great applicability and effectiveness for real-time applications. We plan to optimize the latency upper bounds, address the implementation complexity of wormhole-based SHARP and explore the more efficient real-time NoCs in our future work.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China (NSFC No. 61772094), the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (MOE2019-T2-1-071) and Tier 1 (MOE2019-T1-001-072), and Nanyang Technological University, Singapore, under its NAP (M4082282) and SUG (M4082087).

References

[1] B. Kim and et al, "A Real-time Communication Method for Wormhole Switching Networks," in *ICPP*. IEEE, 1998, pp. 527–534.
[2] Z. Shi and et al, "Real-time Communication Analysis for On-chip Networks with Wormhole Switching," in *NOCS*. IEEE, 2008, pp. 161–170.
[3] H. Kashif and et al, "Using Link-level Latency Analysis for Path Selection for Real-time Communication on NoCs," in *ASP-DAC*. IEEE, 2012, pp. 499–504.
[4] X. Zhao and et al, "Per-flow Delay Bound Analysis Based on a Formalized Microarchitectural Model," in *NOCS*. IEEE, 2013, pp. 1–8.
[5] H. Kashif and et al, "SLA: A Stage-level Latency Analysisfor Real-time Communicationin a Pipelined Resource Model," *TC*, vol. 64, no. 4, pp. 1177–1190, 2014.
[6] Q. Xiong and et al, "Real-time Analysis for Wormhole NoC: Revisited and Revised," in *GLSVLSI*, 2016, pp. 75–80.
[7] B. Nikolic and et al, "A Tighter Real-time Communication Analysis for Wormhole-switched Priority-preemptive NoCs," *arXiv*, 2016.
[8] L. S. Indrusiak and et al, "Analysis of Buffering Effects on Hard Real-time Priority-preemptive Wormhole Networks," *arXiv*, 2016.
[9] S. Tobuschat and et al, "Real-time Communication Analysis for Networks-on-Chip with Backpressure," in *DATE*. IEEE, 2017, pp. 590–595.
[10] Q. Xiong and et al, "Extending Real-Time Analysis for Wormhole NoCs," *TC*, vol. 66, no. 9, pp. 1532–1546, 2017.
[11] L. S. Indrusiak and et al, "Buffer-aware Bounds to Multi-point Progressive Blocking in Priority-preemptive NoCs," in *DATE*. IEEE, 2018, pp. 219–224.
[12] B. Nikolić and et al, "Real-time Analysis of Priority-preemptive NoCs with Arbitrary Buffer Sizes and Router Delays," *Real-Time Systems*, pp. 1–43, 2018.
[13] T. Krishna and et al, "Breaking the On-chip Latency Barrier Using SMART," in *HPCA*. IEEE, 2013, pp. 378–389.
[14] C.-H. O. Chen and et al, "SMART: A Single-cycle Reconfigurable NoC for SoC Applications," in *DATE*. IEEE, 2013, pp. 338–343.
[15] T. Krishna and et al, "SMART: Single-cycle Multihop Traversals over a Shared Network on Chip," *micro*, vol. 34, no. 3, pp. 43–56, 2014.
[16] Y. Asgarieh and et al, "Smart-Hop Arbitration Request Propagation: Avoiding Quadratic Arbitration Complexity and False Negatives in SMART NoCs," *TODAES*, vol. 24, no. 6, p. 64, 2019.

[17] I. Pérez and et al, "SMART++: Reducing Cost and Improving Efficiency of Multi-hop Bypass in NoC Routers," in *NOCS*. ACM, 2019, p. 5.

[18] X. Chen and et al, "Reducing Wire and Energy Overheads of the SMART NoC Using a Setup Request Network," *TVLSI*, vol. 24, no. 10, pp. 3013–3026, 2016.

[19] B. K. Daya and et al, "Quest for High-Performance Bufferless NoCs with Single-cycle Express Paths and Self-Learning Throttling," in *DAC*. IEEE, 2016, pp. 1–6.

[20] L. Yang and et al, "Task Mapping on SMART NoC: Contention Matters, Not the Distance," in *DAC*, 2017, pp. 1–6.

[21] W. Liu and et al, "Work-in-Progress: Fixed Priority Scheduling of Real-Time Flows with Arbitrary Deadlines on SMART NoCs," in *EMSOFT*. IEEE, 2017, pp. 1–2.

[22] P. Chen and et al, "Contention Minimized Bypassing in SMART NoC," in *ASP-DAC*. IEEE, 2020, pp. 205–210.

[23] K. G. Harteros and et al, "Fast Parallel Comparison Circuits for Scheduling," *Institute of Computer Science, FORTH*, 2002.

[24] N. E. Jerger and et al, "On-Chip Networks," *Synthesis Lectures on Computer Architecture*, vol. 12, no. 3, pp. 1–210, 2017.

[25] W. J. Dally and et al, *Principles and Practices of Interconnection Networks*. Elsevier, 2004.

[26] C. L. Liu and et al, "Scheduling Algorithms for Multiprogramming in A Hard-Real-Time Environment," *JACM*, vol. 20, no. 1, pp. 46–61, 1973.

[27] Z. Shi and et al, "Priority Assignment for Real-time Wormhole Communication in On-chip Networks," in *RTSS*. IEEE, 2008, pp. 421–430.

[28] E. A. Rambo and et al, "Worst-case Communication Time Analysis of Networks-on-Chip With Shared Virtual Channels," in *DATE*, 2015, pp. 537–542.

[29] M. Liu and et al, "Tighter Time Analysis for Real-Time Traffic in On-Chip Networks with Shared Priorities," in *NOCS*, 2016.

[30] T. Wu and et al, "Analysing Real-Time Traffic in Wormhole-Switched On-Chip Networks," 2016.

[31] Z. Shi and et al, "Improvement of Schedulability Analysis with a Priority Share Policy in On-Chip Networks," 2009.

[32] M. Liu and et al, "A Dependency-graph Based Priority Assignment Algorithm for Real-time Traffic over NoCs with Shared Virtual-Channels," in *WFCS*. IEEE, 2016, pp. 1–8.

[33] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in *RTSS*. IEEE, 1990, pp. 201–209.

[34] Y. Hoskote and et al, "A 5-GHz Mesh Interconnect for A Teraflops Processor," *Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[35] T. Krishna and et al, "SWIFT: A SWing-Reduced Interconnect For a Token-Based Network-on-Chip in 90nm CMOS," in *ICCD*. IEEE, 2010, pp. 439–446.

[36] S. Park and et al, "Approaching the Theoretical Limits of a Mesh NoC with A 16-Node Chip Prototype in 45nm SOI," in *DAC*. ACM, 2012, pp. 398–405.

Peng Chen received the B.E. degree from the School of Big Data & Software Engineering at Chongqing University, Chongqing, China, in 2015 and is currently pursuing the Ph.D. degree from the College of Computer Science at Chongqing University, Chongqing, China. He is now a visiting scholar at the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His current research interests include embedded and real-time systems, on-chip multi-core systems, real-time scheduling, and network-on-chip.

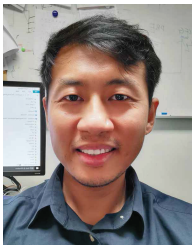
Weichen Liu is a Nanyang Assistant Professor at the School of Computer Science and Engineering, Nanyang Technological University, Singapore. He received the Ph.D. degree from the Hong Kong University of Science and Technology, and the BEng and MEng degrees from Harbin Institute of Technology, China. Dr. Liu authored and co-authored more than 100 research papers in peer-reviewed journals, conferences, and books. His research interests include embedded and real-time systems, multiprocessor systems, network-on-chip, and machine learning acceleration.

Hui Chen received the B.E. degree from the School of Computer Science and Technology at Zhejiang University, China, and the M.S.c degree from the School of Computing at National University of Singapore, Singapore. She is currently pursuing the Ph.D. degree from the School of Computer Science and Engineering, Nanyang Technological University, Singapore. Her current research interests include machine learning and network-on-chip.

Shiqing Li received the B.S. and M.S. degree in computer science and technology from Shandong University, Jinan, China, in 2016 and 2019. He is currently pursuing the Ph.D. degree from the School of Computer Science and Engineering at Nanyang Technological University, Singapore. His current research interests include FPGAs, deep learning, and high-level synthesis.

Mengquan Li received the B.E. degree from Chongqing University in 2015 and is currently pursuing the Ph.D. degree from the College of Computer Science at Chongqing University, China. She is now a visiting scholar in the School of Computer Science and Engineering, Nanyang Technological University, Singapore. Her current research interests include optical network-on-chip, on-chip temperature modeling and optimization, and nanophotonic accelerator design for deep learning.

Lei Yang received her Ph.D. degree and B.E. degree in 2019 and 2013 from Chongqing University, Chongqing, China. She is currently a Postdoc Researcher, and will join as an Assistant Professor in the Department of Electrical Computer Engineering at the University of New Mexico, NM, USA. Dr. Yang's research interests are in automated machine learning, embedded systems and high-performance computing architectures.



Nan Guan is currently an Associate Professor at the Department of Computing, The Hong Kong Polytechnic University. His research interests include real-time embedded systems, cyber-physical systems (CPS) and Internet-of-Things (IoT). He received the ACM SIGBED Early Career Researcher Award in 2020, the EDAA Outstanding Dissertation Award in 2014, the Best Paper Award of RTSS 2009, the Best Paper Award of DATE 2013, the Best Paper Award of ACM e-Energy 2018, the Best Paper Award of ISORC 2019, the Outstanding Paper Award of RTSS 2019. He is the TPC Chair of RTAS 2021, Deputy Track Chair of RTSS 2020, Track Co-Chair of ICPADS 2020, Program Co-Chair of SETTA 2019, Track Chair of RTAS 2018, Program Co-Chair of ICESS 2017 and Program Co-Chair of EMSOFT 2015.