On the Capacity of Bufferless Networks-on-Chip

Alexander Shpiner, Erez Kantor, Pu Li, Israel Cidon and Isaac Keslassy
Department of Electrical Engineering,
Technion - Israel Institute of Technology, Haifa 32000, Israel
{shalex@tx, erez.kantor@ee, pu.li@tx, cidon@ee, isaac@ee}.technion.ac.il

***Abstract*—Networks-on-Chip (NoCs) form an emerging paradigm for communications within chips. In particular, bufferless NoCs require significantly less area and power consumption, but also pose novel major scheduling problems to achieve full capacity.**

In this paper, we provide first insights on the capacity of bufferless NoCs. In particular, we present *optimal* periodic schedules for several bufferless NoCs with a complete-exchange traffic pattern. These schedules particularly fit distributed-programming models and network congestion-control mechanisms. In addition, for general traffic patterns, we also introduce efficient greedy scheduling algorithms, that often outperform simple greedy online algorithms and cannot have deadlocks. Finally, using network simulations, we quantify the performance gain of our suggested algorithms, and show how they improve throughput by up to 35% on a torus network.

I. Introduction

A. Background

Networks-on-Chip (NoCs) form an emerging paradigm for communications within large VLSI systems implemented on a single silicon chip. In a NoC system, modules such as processor cores, memories and specialized blocks exchange data using a network, rather than simple shared busses as in previous systems. A NoC is constructed from multiple point-to-point data links interconnected by routers, such that messages can be relayed from any source module to any destination module over several links, by making routing decisions at the switches.

Interestingly, despite the revolution that the NoC paradigm is causing in computer architecture, *little is known on the capacity region of NoCs*. This is especially surprising because any slight improvement in the capacity of NoCs may have a huge impact. For instance, NoCs are present in most personal computers currently sold around the world [1].

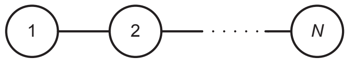
There are many possible NoC topologies, as shown in Figure 1. They include simple *line* and *ring* topologies [2], which are widely used in optics-based networks [3]–[5]. For instance, the Intel Sandy Bridge CPU [1], and the IBM Cell Broadband Engine [6] use a ring-based interconnect for their on-chip-network. The *mesh* and the *torus* NoC topologies are also popular [7], [8]. In this paper, we study these four main topologies: line, ring, mesh and torus.

We are especially interested in NoCs that (a) use *bufferless* switches, and (b) carry a *periodic* traffic. First, *bufferless* NoCs are particularly interesting because they offer a better area and power performance than those with buffers, in exchange for an increased scheduling complexity. In fact, NoC buffers can consume significant dynamic and static energy, and occupy a large chip area [9]–[12].

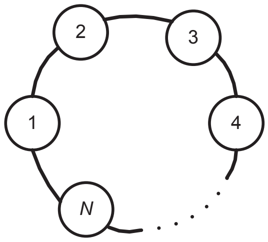
Second, we look at applications with a *periodic* traffic. Time-critical periodic applications for embedded devices combine ever-growing computing demands with hard-deadline performance-guarantee requirements. These applications, such as distributed-programming models and network congestion control mechanisms, use a periodic pre-determined traffic pattern with tight deadlines. But the embedded devices often cannot provide the required hard-deadline performance-guarantees. This is because they often rely on NoCs with best-effort communication, which results in an unpredictable network behavior, causing a significant application performance variability.

Our goal is to devise a periodic schedule algorithm that can provide a capacity-optimal guaranteed service for this traffic pattern in bufferless NoCs. We also later compare such scheduling algorithms that use the pre-determined nature of the periodic schedule with greedy online algorithms that would not rely on this assumption. We introduce a novel theoretical model of bufferless NoC architectures, and find periodic conflict-free schedules. This is illustrated in the next simple example.

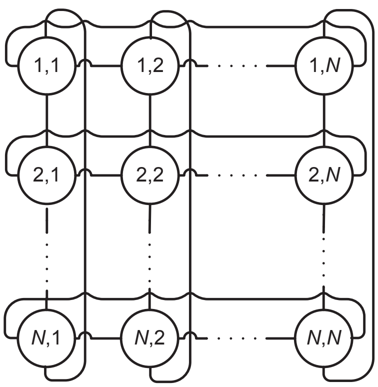
Example 1. *Consider a simple case presented in Figure 2, where the bufferless NoC sub-network consists of three nodes A, B and C and two links 1 and 2 connecting them. The traffic requirement consists of three packets that need to be sent at each period : $A \to B$, $B \to C$ and $A \to C$. The periodic schedule is created by setting for each packet the time slot at which it is sent in each period. Assume equal propagation times on each link, such that each time slot is sufficient to send a packet on a link. Assume first that the transmissions are scheduled*



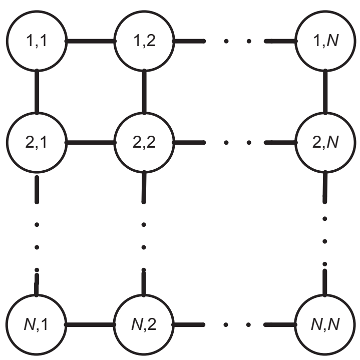
(a) Line.



(b) Ring.



(c) Torus.



(d) Mesh.

Fig. 1. NoC topologies.

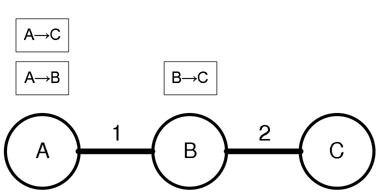


Fig. 2. Simple case with a three-node bufferless NoC sub-network.

TABLE I
EXAMPLE OF NON-OPTIMAL SCHEDULE

time slot	link 1	link 2
1	$A \to B$	$B \to C$
2	$A \to C$	
3		$A \to C$

TABLE II
EXAMPLE OF OPTIMAL SCHEDULE

time slot	link 1	link 2
1	$A \to C$	$B \to C$
2	$A \to B$	$A \to C$

online as shown in Table I. In this example, $A \to B$ is scheduled in the first time slot. Next, packet $B \to C$ is also scheduled in the first time slot, because link 2 is free. Finally, $A \to C$ is scheduled in the second time slot. It takes two slots to complete the transmission of $A \to C$. Therefore, the length of the total schedule is 3 time slots. As shown in Table II, the above schedule can be optimized. In the first time slot packet $A \to C$ is scheduled on link 1, and in the second time slot on link 2. Link 2 is free in the first time slot, therefore, packet $B \to C$ can be scheduled. Finally, packet $A \to B$ is scheduled in the second time slot on link 1. The links are utilized all the time, therefore, this schedule is optimal and reaches its full capacity. Its length is 2 time slots.

Note that if the network were buffered, then of course, there would be no need for collision-free scheduling. The collided packets could then be queued at the buffers. However, the buffers would lead to significant power and area costs.

Our suggested periodic transmission of scheduled packets over the NoC can be complemented by best-effort packet services at a strictly lower priority. Therefore, our scheduling algorithms can be used to provide a two-tier solution for both guaranteed-service and best-effort communications in a NoC.

B. Applications

Many applications and parallel computation models rely on a predetermined periodic traffic. We focus in this paper on a periodic *uniform complete-exchange* communication pattern and on its optimal schedule in different topologies. This uniform complete-exchange pattern, also known as all-to-all personalized communication, or scatter-gather pattern, is a type of collective communication pattern for Message Passing Interface (MPI) [13], in which all processors need to communicate with all other processors. In this communication pattern, each of the N processors in the network has a distinct, but equal-size, message to send to each of the remaining $N-1$ processors. It is thus a highly dense communication pattern that can result in many link contentions [14]–[19]. This simple communication pattern enables many numerical parallel algorithms. For example, it includes an increasingly large class of parallel algorithms, such as neural networks, large FFT (Fast Fourier Transform) computations, parallel quicksort, matrix transpose, array redistribution, distributed table lookup and bitonic sort [20]. More generally, any multi-round algorithm where updates are sent to many randomly-chosen processing elements can be enabled using a topology that allows uniform communications with updates sent to all other processing elements.

It is interesting to note that numerous applications have been shown to allow restructuring that completely separates *the computation logic* from *the communication module*. Such structure complies to a well-established programming model called *Bulk Synchronous Parallel (BSP)* [21]. In BSP, programs are represented as a series of two *super-steps*: first, a computation super-step, then, a communication super-step. BSP has gained popularity for its ability to enable predictable performance on a variety of parallel and distributed platforms, from super-computers to CMPs. Compilers can apply profile-based or static code analysis to determine the pattern in the communication super-step of a BSP application, precompute the communication schedule for a given NoC topology, and augment the compiled code with this information to allow its use at runtime.

The uniform complete-exchange communication pattern could also fit many *end-to-end congestion control* mechanisms in which all nodes exchange information at regular intervals. Such mechanisms can include the estimation of network delays and/or losses in the best-effort network; the periodic acknowledgment of received packets; a generalized hot-spot rate and fairness management; as well as a source-destination queue management [22].

C. Contributions

The main contribution of this paper is the introduction of *capacity-optimal periodic schedules for uniform traffic over bufferless NoCs, under several topologies*.

The optimality of the schedule is on the capacity utilization of the links, or in other words, the length of the period. We also leverage the fact that the traffic pattern is fixed to compute the schedule only once offline (for instance just after the application compilation). Therefore, by using an optimal offline schedule, we aim to provide a higher capacity utilization, or a shorter period, and hence transmit the same traffic demands with higher throughput using the same bufferless NoC architecture. Using this optimization framework, we prove the existence of several optimal scheduling algorithms on different NoC topologies.

First, we present an algorithm, named Algorithm *DTNS* (Degree-Two NoC Scheduling), for complete-exchange communication in degree-two networks, e.g., line and ring NoC topologies. We prove its optimality and also provide several results on its period length.

Second, we present an algorithm, named Algorithm *TNS* (Torus NoC Scheduling), for complete-exchange in $N \times N$ torus NoC topologies, and prove its optimality. We later provide lower and upper bounds on the performance of any minimal schedule in the mesh NoC topology. We also provide a constant bound on the ratio between the performance of the TNS algorithm in a mesh and its optimal performance in a torus.

In addition, we introduce a greedy latency-based scheduling algorithm for a general traffic pattern. In this case the input to our scheduling mechanism is the set of flows with their bandwidths and pre-computed routes [23]–[25]. Then our algorithm allocates the time-slots to the flows in order to minimize the length of the schedule period. This schedule provides both guaranteed throughput and guaranteed latency while using bufferless routers. Note that it could also be used in congestion-free NoCs like optical NoCs; this is, however, beyond the scope of the paper. Finally, using simulations, we show that the proposed schedules outperform approximations of previously-known online algorithms, typically by over 20% in the ring and torus topologies, and 5-15% in the mesh topologies, depending on the number of nodes.

The rest of the work is organized as follows. We first define the problem in Section III. Then we analyze degree-two topologies in Section IV. We present our proposed algorithm for the torus topology in Section V. In Sections VI and VII we analyze the mesh topology and suggest greedy scheduling algorithms. Last, we present simulation results in Section VIII, before concluding in Section IX.

II. Related Work

The periodic traffic pattern enables us to rely on a *bufferless NoC*. Providing guaranteed service using a bufferless NoC requires finding a periodic conflict-free schedule. Note that not all bufferless NoC designs meet the required real-time guarantee. For example, bufferless NoCs with deflection routing [26], [27] or dropping [28], [29] are often characterized by a highly-unpredictable network behavior that does not fit our requirements. In fact, [27] shows that buffered VC NoCs have a better performance, lower cost and complexity than bufferless NoCs with deflection. However, in this paper, we consider periodic collision-free scheduling that is predetermined and offline, and therefore, does not waste resources on longer routes as in deflection routing. Recent work [30] identified the importance of congestion control in bufferless NoCs and presented source throttling-based mechanism to reduce congestions. A bufferless NoC architecture that provides guaranteed service was introduced in Aethereal [31]–[33]. The Aethereal architecture relies on a greedy resource-reservation algorithm that is designed to adapt to changing traffic patterns. In particular, the UMARS algorithm relies on an offline scheduling by ordering the flows by their bandwidth requirements prior to scheduling them greedily by shortest and less contented route [23]. Other works suggest algorithms for route optimization with multiple paths and shortest latency by keeping in-order arrivals in offline [24], [34] and online [25], [35] modes. However, all these works do not explicitly attempt to *achieve the network capacity* and do not provide an optimal solution for the period length. These papers can nicely extend our work on non-uniform traffic by providing efficient algorithms for routing and mapping, which we do not consider in this paper.

In addition, there have been other works on providing guaranteed service in buffered NoCs. Most suggested architectures, like Nostrum, have relied on router buffers to provide guaranteed service, for instance by using temporally-disjoint networks [36], [37]. Slot allocation in a TDM network-on-chip was introduced in [38]. However it assumes the existence of buffers and virtual channels in the routers, and does not attempt to provide an optimal scheduling. Another common approach relies on bounding the traffic rate using a leaky-bucket framework, while still relying on wormhole routing [39]. The packet injection into the network is controlled and bounded to preserve quality-of-service. Additional papers also focus on collective communication patterns in the NoCs [40]–[42]. However, they all assume wormhole routing and buffered NoC. Finally, it is also possible to provide statistical instead of deterministic guarantees [43].

An alternative to TDM is SDM, which divides the links by space and not by time. [44] proposes a communication scheme by dividing the resources of a traditional packet-switched NoC between a packet-switched and a circuit-switched sub-networks. Another way to construct congestion-free network is to use simple structures such as rings. [45] presents a network design that is based on the construction of multiple virtual rings and provides scalable aggregate throughput and absence of packet loss. Further, [46] introduces design principles of a ring network with spatial bandwidth reuse.

From the theory perspective, bufferless routing was intensively studied under various of names and models. For example, *Direct Routing* [47] defines the problem such that for a given set of packets with corresponding source and destination, the objective is to schedule the injections times of the packets. In different versions of the problem the specified path can be given as an input to the algorithm or defined as another output. The solution requirement is to avoid collisions, and to minimize the schedule period time. [47] presents a randomized $O(d^2 \log^2 n)$-approximation algorithm for d-dimensional Mesh for general traffic pattern. They show that finding an optimal Direct Routing is NP-hard, and for general networks, they show that this problem cannot be easily approximated. Also, in unbuffered optical networks, [48] considers how to schedule packets optimally given several wavelengths.

Another related problem is *hot-potato-routing* studied in [49]–[51], where for a given set of packets, where each packet consists of a source vertex, a destination vertex and an injection time. However, the packets are deflected from their shortest route, therefore the provided solution is not optimal. In addition, packet scheduling in bufferless linear networks was investigated before in offline [52] and online [53] versions. However, no assumption on uniformity or periodicity of the traffic was considered, therefore they could not provide an optimal solution. The baked potato routing algorithm [54] was also among the first to consider switch scheduling in a bufferless network for periodic traffic. However, it only provided a solution for a spanning tree network.

III. Problem Definition

In the paper we investigate a bufferless network-on-chip architecture in which the traffic is produced under some periodic traffic demand pattern. The objective is to find a *periodic schedule* Δ with *minimal period length* S needed to service traffic pattern λ, i.e. to maximize the capacity utilization. The periodic schedule Δ has to provide congestion-free routing, so that each packet reaches its destination using a *shortest* route.

Formally, consider a network of n nodes. Each node is composed of a router connected to up to 4 neighbor nodes and to a single local module (traffic producer/consumer unit). *The nodes do not buffer, drop or deflect packets.*

Assume that the traffic is produced under some periodic traffic demand pattern. Denote as *uniform* or *complete exchange* a traffic pattern in which each local module of a node, during each period, injects exactly one packet destined to each local module of other nodes in the network. In other words, the normalized traffic demand pattern is $\lambda_{i \to j} = 1, \forall i, j$, where i and j are nodes in the network. Further assume that all link capacities are equal to a normalized unit capacity, equivalent to sending one unit-sized packet per time-slot, and denote by C_{tot} the total link capacity, i.e. the number of links in the network.

Note that larger packets can be subdivided into the unit-size packets that we consider, and which are also sometimes called flits. Then, multi-flit packets can be sent over several schedule periods. These flits can be queued in the nodes before being injected into the network, but once they enter the network, they are not queued in any router before reaching their destination. In the paper, we will use the generic term of *packets* to denote these unit-sized flits.

At each time slot, each router can send a single packet to each adjacent router or to its associated module.

Therefore, each router can receive up to 5 packets (from the up-to 4 adjacent routers and the module), and *must* forward them immediately at the next slot. Note that since the schedule is periodic and pre-determined, there are no deadlocks and no livelocks. Finally, we say that an algorithm A is *optimal*, if A produces a minimal period length schedule for the complete-exchange traffic requirements.

IV. Optimal Algorithm for Complete Exchange in Degree-Two NoCs

A degree-two NoC is a NoC topology in which all nodes have a degree of either one or two. There are two types of degree-two NoCs. We denote by *n-Line* a NoC consisting of n nodes connected by $n-1$ bi-directional links in a line topology, as illustrated in Figure 1(a); and by *n-Ring* a NoC consisting of n nodes connected by n bi-directional links in a ring topology, as illustrated in Figure 1(b).

A. DTNS Scheduling Algorithm

We start by designing a collision-free scheduling algorithm for complete-exchange in degree-two networks, called *DTNS* (Degree-Two NoC Scheduling). Define a ℓ-hopped packet as a packet whose source-to-destination distance in the shortest route is ℓ links. The algorithm is built in the following way. Each node i at each time slot t operates according to the scheme:

- If in time slot $t-1$ the node receives a packet for forwarding, then it forwards it at time t.
- Otherwise, it starts transmitting an ℓ_{max} -hopped packet, where ℓ_{max} is the largest number of hops ℓ of all packets left with the node i as their source node.

As an example, note that DTNS for $n=3$ produces the schedule presented in Table II. We obtain the following properties of the DTNS algorithm on degree-two networks.

Property 1 (n-Line)**.** *Given an n-Line of n nodes, the schedule period length $S_L(n)$ of the DTNS-based schedule is:*

$$S_L(n) = \begin{cases} \frac{n^2}{4} & \text{if } n \text{ is even;} \\ \frac{n^2-1}{4} & \text{if } n \text{ is odd.} \end{cases} \quad (1)$$

Proof: The traffic in the n-Line network can be separated into two distinct groups, one for each direction. Therefore, for simplicity, we consider only a single direction in the analysis. By the rules of a DTNS algorithm the packet from node 1 to node n ($1 \to n$), as presented in Figure 1(a), with propagation time $n-1$ slots is transmitted first. It is easy to see that during its transmission, all the packets $1 \to a$ and $a \to n$, where $a = 2, \ldots, n-1$ are transmitted. Next, packet $2 \to n-1$ is transmitted in $n-3$ time slots, during which all the packets $2 \to a$ and $a \to n-1$ where $a = 3, \ldots, n-2$ are transmitted. Summing over all successive packets times, we directly obtain the following schedule length:

$$S_L(n) = \begin{cases} \sum_{i=0}^{\frac{n}{2}-1} (n-2i-1) = \frac{n^2}{4} & \text{if } n \text{ is even;} \\ \sum_{i=0}^{\frac{n-1}{2}-1} (n-2i-1) = \frac{n^2-1}{4} & \text{if } n \text{ is odd.} \end{cases}$$

■

Property 2 (n-Ring)**.** *Given an n-Ring of n nodes, the schedule period length $S_R(n)$ of the DTNS-based schedule is*

$$S_R(n) = \begin{cases} \frac{n^2}{8} & \text{if } n \text{ is even;} \\ \frac{(n-1)(n+1)}{8} & \text{if } n \text{ is odd.} \end{cases} \quad (2)$$

Proof Outline: The proof is very similar to the proof of Property 1, and consists of summing over all successive packet times. We then obtain:

$$S_R(n) = \begin{cases} \sum_{i=1}^{\frac{n}{2}-1} i + \frac{n}{4} = \frac{n^2}{8} & \text{if } n \text{ is even;} \\ \sum_{i=1}^{\frac{n-1}{2}} i = \frac{(n-1)(n+1)}{8} & \text{if } n \text{ is odd.} \end{cases}$$

■

Note that the above property assumes that consecutive periods might overlap. In other words, the transmissions of a new period can be started before the last period was ended. Otherwise, if overlapping is forbidden, we obtain a worse result for even n's. This is because overlapping enables an alternate routing of $\frac{n}{2}$-hop packets: one period in clockwise direction, and the next period in counterclockwise direction. Without overlap, we obtain the following result:

Property 3 (n-Ring without overlap)**.** *Given an n-Ring of n nodes in which period overlapping is forbidden, the schedule period length $S_{R,no}(n)$ of the DTNS-based schedule is*

$$S_{R,no}(n) = \begin{cases} \frac{n(n+2)}{8} & \text{if } n \text{ is even;} \\ \frac{(n-1)(n+1)}{8} & \text{if } n \text{ is odd.} \end{cases} \quad (3)$$

Proof Outline: The proof is similar again to the proofs above, and simply consists of summing over all successive packet times. We then obtain:

$$S_{R,no}(n) = \begin{cases} \sum_{i=1}^{\frac{n}{2}} i = \frac{n(n+2)}{8} & \text{if } n \text{ is even;} \\ \sum_{i=1}^{\frac{n-1}{2}} i = \frac{(n-1)(n+1)}{8} & \text{if } n \text{ is odd.} \end{cases}$$

■

The DTNS algorithm gives a higher priority to the retransmitted packets over new packets at each node, and thus avoids congestions on several types of NoC topologies, as formulated in the next theorem.

Theorem 4. *The DTNS algorithm is optimal on n-Lines and n-Rings with complete-exchange traffic.*

Proof: The proof is based on the fact that the bottleneck links are always utilized in the N-Line, and likewise that all the links are always utilized in the n-Ring (they are all bottleneck links). As a consequence, no other schedule can be more efficient using a smaller period length S.

First we prove optimality on the n-Line. We consider two cases.

Case 1: $n = 2k + 1$ is odd. Let $l_{mid} = (k, k+1)$ be the directed link that connects between node k and node $k+1$. It is easy to verify that l_{mid} requires to deliver $k \cdot (k+1) = \frac{n^2-1}{4}$ packets, which is clearly a lower bound of the period length.

Case 2: $n = 2k$ is even. The number of packets that l_{mid} requires to deliver is $(k/2)^2 = n^2/4$.

Next, we prove optimality on the n-Ring. Again, we consider two cases.

Case 1: $n = 2k + 1$ is odd. It is easy to see that each link transmits i times i-hopped packet, when $i = 1, \ldots, k$. Summing the transmission of all the packets we get the length of the optimal period $S(n) = \frac{(n-1)(n+1)}{8}$ which is equal to the result in Equations (2) and (3).

Case 2: $n = 2k$ is even. The k-hopped packets have two shortest paths. Therefore, in each period k-hopped packets are transmitted only in one direction, while the links on the another direction are idle. Therefore the length of the period consists of the time for the transmission of i-hopped packets ($i = 1, \ldots, k-1$), which is equal to $\sum_{i=1}^{k-1} i$ time slots, and the time for the the transition of the k-hopped packets, which is equal to k time slots. Summing above, we get the result in Equation (3). Of course, this result is not optimal, because of the idle links. If overlapping of the scheduling between the two adjacent periods is allowed, then the idle links can used to transmit the k-hopped packets of the next period, and thus the average number of time slots to schedule the k-hopped packets is equal to $\frac{k}{2}$. Thus, we get the result in Equation (2). ∎

V. Optimal Algorithm for Complete Exchange in Torus NoCs

In this section, we present an algorithm, named TNS (*Torus NoC Scheduling*), which is designed to provide a periodic schedule over the Torus NoC topology (Figure 1(c)) for the complete-exchange traffic pattern. We will demonstrate that TNS is guaranteed to achieve the network capacity in an $N \times N$ Torus network with $n = N^2$ nodes, using the same setting assumptions of uniform traffic and unit capacities.

The TNS algorithm provides the injection time to the network and the minimal-length route for each packet within the period, based on the source and destination of the packet. It also guarantees that there are no packet collisions. Therefore, *the TNS algorithm does not rely on buffering or dropping packets, and it also has no deadlocks*.

A. TNS Algorithm Description

Consider an $N \times N$ torus network with $n = N^2$ nodes, and total allowed capacity C_{tot} for all links in the network. Each node is composed of a router connected to up to 4 neighbor nodes and to a single local module (traffic producer/consumer unit). Denote the nodes as a set of tuples $\{(x, y) \mid x, y \in \{1, ..., N\}\}$, where the first entity refers to the rows on the torus and the second entity refers to the columns on the torus. Denote by $DIST(a, b) = \min\{\ |a-b|\ ,\ N-|a-b|\ \}$, the *distance* between a and b, for every $a, b \in \{1, ..., N\}$.

We now present the TNS algorithm in detail. Intuitively, TNS decomposes the period of length S into several *phases* of unequal lengths. In each phase, it transmits all the packets between all source-destination pairs that have the same maximum distance across dimensions. TNS further decomposes each phase into several sub-phases called *epochs*. In each epoch, it connects nodes that are not only at the same maximum distance, but that also follow a given pattern. We now define the TNS algorithm more formally.

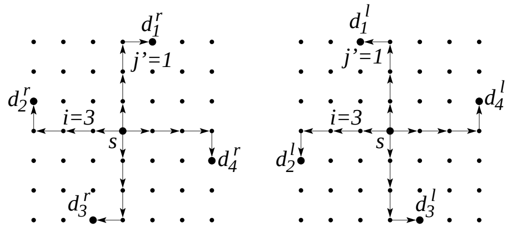
Phases — The schedule period is divided into $\lfloor N/2 \rfloor$ *phases*. A packet belongs to the *envelop* of a square of nodes $(i+1) \times (i+1)$, if the maximum distance across all dimensions between its source and its destination is equal to i. In *phase* i, packets in all the envelops of squares of size $(i+1) \times (i+1)$ are scheduled to be transmitted.

Epochs — Phase i consists of $2i$ *epochs* for $1 < i < N/2$. If N is even, phase $N/2$ consists of $i+1$ epochs. On epoch $j \in \{0, 1, ..., 2i-1\}$ of phase i, each node transmits four packets to four different destinations by crossing exactly $i + j'$ links, where $j' = j$, if $j \leq i$ and otherwise, $j' = j - i$. First i links in one direction and the other j' links in the perpendicular direction, as explained below. The destinations are given by the following three steps, for each of the four packets, one for each direction, as illustrated in Fig. 3:

1) First, the packet follows a direct traversal walk on i links;
2) Then, in epoch 0 it stops; in the epoch $(j|0 < j \leq i)$, it "turns right" in a clockwise direction; in the epoch $(j|i < j < 2i)$, it "turns left" in counter-clockwise direction.
3) Finally, there is another direct traversal walk through j' additional links.

Thus, epoch j of phase i takes $i + j'$ time-slots. Finally, after completing all the $N/2$ phases, all the pairs of nodes in the Torus have been connected and have transmitted a packet exactly once.

Now that we have formally defined TNS, we show in which epoch each packet is transmitted. First, for a given source node (a, b) and destination node (c, d), denote by $(a, b) \rightarrow (c, b) \rightarrow (c, d)$ a shorter path that goes from (a, b) to (c, d) via (c, b). If $DIST(a, c) \geq DIST(b, d)$,



(a) (b)

Fig. 3. Two epochs in phase $i = 3$. (a) illustrates epoch $j = 1$, with the traversal walk and a turn in a clockwise direction. (b) illustrates epoch $j = 3 + 1 = 4$, with the traversal walk and a turn in a counter-clockwise direction.

then we say that this path is a *long-then-short shorter* path. (Note that the TNS algorithm uses only long-then-short shorter paths.) The algorithm sets a unique phase i and epoch j for each source-destination pair, denoted by the pair (i, j). For instance, consider a packet from source node (a, b) to destination node (c, d). The *phase* i in which the packet is transmitted is given by:

$$i = \max \{DIST(a, c), DIST(b, d)\}.$$

Moreover, the *epoch* in which the packet is transmitted is either $j = j^*$ or $j = j^* + i$, where

$$j^* = \min \{DIST(a, c), DIST(b, d)\}.$$

The exact epoch is decided by the direction (clockwise or counter-clockwise) of the traversal walk of a long-then-short shorter path between source node (a, b) and destination node (c, d). If there exist two different long-then-short shorter paths (i.e., for the case where $DIST(a, b) = DIST(c, d)$, that is $i = N/2$ for even N), then the packet is delivered on epoch $j = j^*$.

Lemma 1 (TNS length)**.** *Given an $n = N \times N$ Torus, the length of the schedule period $S_T(n = N \times N)$ of the TNS-based schedule is:*

$$S_T(n = N \times N) = \sum_{i=1}^{\lfloor N/2 \rfloor} \left(2 \sum_{j=i}^{2i} j - 3i \right) = \frac{N^3 - N}{8} = \frac{n\sqrt{n} - \sqrt{n}}{8}, \quad (4)$$

for odd N and

$$S_T(n = N \times N) = \frac{N^3 + 2N}{8} = \frac{n\sqrt{n} + 2\sqrt{n}}{8}$$

for even N. Furthermore, when overlap is forbidden, the result for even N becomes

$$S_T(n = N \times N) = \frac{N^3}{8} + N = \frac{n\sqrt{n}}{8} + \sqrt{n}$$

Proof: Consider an odd N. For $N \times N$ Torus, all connections between all the nodes are covered within $\lfloor N/2 \rfloor$ *phases*. It is the maximum distance across all dimensions between the sources and the destinations of all the packets.

The shortest *epoch* in the *phase* i is the one that transmits packets in one dimension, i.e., $c = a$ or $d = b$. Its length is i time slots, because packets with latency of i hops are scheduled within it. The longest *epoch* in the *phase* i is the one that transmits packets between the corners of the $(i + 1) \times (i + 1)$ square ($DIST(a, c) = DIST(b, d)$). Its length is $2i$ time slots. Other *epochs* are scheduled twice, one for each *direction*. Therefore, *phase* i consists of one *epoch* for one-dimensional packets, one *epoch* for corner packets and two *epochs* for other packets, one in each *direction*. Thus, the length of phase i is

$$S_{T,i}(n = N \times N) = 2 \sum_{j=i}^{2i} j - 3i = 3i^{\ 2} \quad (5)$$

time slots, for every $i < N/2$. For even N, the last phase (phase $N/2$) has only $N/2 + 1$ epochs that takes $\sum_{j=N/2}^{N} j$ time slots. It is simply counted differently whether or not there is overlap. ∎

Theorem 5. *The TNS algorithm is optimal on Torus NoCs topology with complete-exchange traffic.*

Proof: Depending on whether N is odd or even. These two cases are considered separately.

Case 1: N is odd: Denote $L_{(i,j)\to(k,l)}$ as the minimum number of hops that a packet passes from node (i, j) to node (k, l) $(i, j, k, l \in \{1, ..., N\})$, and it is given as the sum of the distances on both axes:

$$L_{(i,j)\to(k,l)} = DIST(i, k) + DIST(j, l). \quad (6)$$

For the uniform traffic, the total number of hops for all packets in a period $S_t(N)$ is given by:

$$L_t = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} \sum_{l=1}^{N} L_{(i,j)\to(k,l)} = \frac{N^3(N^2 - 1)}{2}. \quad (7)$$

Divide L_t by S_T using Equation (4), and get

$$C_{tot} = L_t/S_T = 4N^2, \quad (8)$$

where C_{tot} is the capacity of the torus, i.e., the total number of available links in the network. It indicates that by this scheduling algorithm, all the links are always utilized during the transmission. Thus, the algorithm is optimal.

Case 2: N is even: Denote *edge packets* as the packets that have two paths with equal distances of $N/2$ in one of the dimensions, for a shortest routing from source node to the destination node. For example, if the equal distance is in the X axis, the source node is (i, k), then the destination node is $((i + N/2) \pmod N), l)$. We first consider the case when *overlapping* is allowed and calculate the transmission of the *edge packets* in this period twice on account of their transmission in the next period. The duration of transmission of the packets that are not the *edge packets* is the duration of the transmission of *phases* 1 to $N/2-1$:

$$S_{\text{t},N/2-1} = \sum_{i=1}^{N/2-1}\left(2\sum_{j=i}^{2i} j - 3i\right) = \frac{N(N-1)(N-2)}{8} \quad (9)$$

time slots. The sum of latencies of the packets that are not the *edge packets* is:

$$L_{\text{t, non-edge}} = = 2N(N-1)\sum_{i}\sum_{k\neq(i+N/2)\bmod N} DIST(i,k) = = \frac{N^3(N-1)(N-2)}{2}. \quad (10)$$

When dividing the sum of latencies $L_{\text{t, non-edge}}$ by the time slots $S_{\text{t},N/2-1}$, we get $4N^2$, which is equal to the total capacity of the links C_{tot}, and indicates that all links are utilized all the time during the transmission. Thus, for the transmission of *non-edge packets*, the scheduling algorithm is optimal. The *edge packets* are transmitted in the phase $N/2$. As we seen before, the *edge packets* have two shortest paths. Therefore, they can be transmitted twice during a single *phase*, each on the one of the two shortest paths. Therefore, the duration of transmission of the *edge packets twice* is the duration of phase $N/2$:

$$S_{N/2} = \left(2\sum_{j=i}^{2i} j - 3i\right)\Bigg|_{i=N/2} = 2\sum_{j=i}^{2i} j - \frac{3N}{2} = \frac{3N^2}{4}. \quad (11)$$

By considering all possible cases, we find that the sum of the latencies of the *edge packets* is:

$$L_{\text{t,edge}} = = 2N^2\sum_{i}\sum_{k=(i+N/2)\bmod N} DIST(i,k) + +2N\sum_{i}\sum_{k} DIST(i,k) = = 2N^2\cdot N\cdot N/2 + 2N\left(N\left(\sum_{d=1}^{N/2-1} 2d + N/2\right)\right) = = N^4 + N^4/2 = \frac{3N^4}{2}. \quad (12)$$

Therefore, the following equality holds:

$$C_{tot} = 4N^2 = \frac{2L_{\text{t,edge}}}{S_{N/2}},$$

which means that transmitting the *edge packets* twice during the phase $N/2$ maximally utilizes the links. One can consider the second transmission of the *edge packets* for the next schedule period. We showed that with uniform traffic and unit link capacities, the links in the Torus are always utilized, hence the algorithm is optimal. ■

VI. Scheduling Bounds for Complete Exchange in Mesh

We now want to provide some intuition on the schedule period in a NoC Mesh topology (Figure 1(d)). In such a topology, the TNS algorithm is not optimal anymore, and we have not found any characterization of a generally optimal algorithm. Therefore, we will provide instead lower- and upper-bounds on the schedule period, and compare it with the torus schedule period.

A. Lower Bound for Mesh Schedule Length

We now establish a lower bound on the period length in the Mesh NoC topology.

Theorem 6. *Given an $n = N\times N$ Mesh, the period length $S_M(n = N\times N)$ of any schedule for complete-exchange traffic satisfies*

$$\begin{cases} S_M(n = N\times N) \geq \frac{N^3}{4} = \frac{n\sqrt{n}}{4} & \text{if } N \text{ is even,} \\ S_M(n = N\times N) \geq \frac{N^3-N}{4} = \frac{n\sqrt{n}-\sqrt{n}}{4} & \text{if } N \text{ is odd.} \end{cases} \quad (13)$$

Proof: As previously, the proof relies on computing the load on the bottleneck links, and using this load for the lower-bound on the period length. In the proof we assume XY routing, and calculate the load on the link that transmits the largest number of packets (the bottleneck link). The lower bound of the period length is the time which is taken to transmit the packets over the bottleneck link.

According to the properties of the XY routing, each horizontal link $((i,j)\rightarrow(i+1,j))$ transmits packets from j nodes $((i,1),(i,2),\ldots,(i,j))$ to $N\cdot(N-j)$ nodes (k,l), where $k = 1,\ldots,N$ and $l = i+1,\ldots,N$.

Thus, the number of packets $P_{i,j\rightarrow i+1,j}$ that are transmitted on link $((i,j)\rightarrow(i+1,j))(N)$ in each period is:

$$P_{i,j\rightarrow i+1,j}(N) = j\cdot N\cdot(N-j) \quad (14)$$

Similar calculations can be obtained for the vertical links.

Equation (14) is maximized on links where $j = \frac{N}{2}$ when N is even and $j = (N-1)/2$ when N is odd, therefore the most utilized link transmits $P_{max}(N)$ packets:

$$P_{max}(N) = \begin{cases} \frac{N^3}{4} & \text{if } N \text{ is even;} \\ \frac{N}{4}\left(N^2-1\right) & \text{if } N \text{ is odd.} \end{cases}$$

Therefore,

$$S_M(n = N\times N) = P_{max}$$

and the result stands. ■

B. Upper Bound for Mesh Schedule Length Using TNS

We now want to provide an upper bound on the period length in a mesh. The application of the TNS algorithm in a mesh topology is as follows. Consider an instance of the TNS algorithm on a $2N \times 2N$ torus. The $N \times N$ mesh is embedded in a $2N \times 2N$ torus by an $N \times N$ subgraph combined from nodes (i, j), where $N/2 < i, j \leq 3N/2$. Using the TNS algorithm on the $2N \times 2N$ torus, the shortest paths between the nodes in this subgraph are routed within the subgraph (contrary to some other shortest paths outside this subgraph that are routed through the boundaries of the torus). For the above $N \times N$ mesh subgraph we use the TNS scheduling of the full $2N \times 2N$ torus, transmitting only the packets between the nodes of the $N \times N$ mesh subgraph, and leaving empty the scheduling slots for the packets outside the subgraph. We can now prove the following result:

Theorem 7. *Let $S_M(n = N \times N)$ denote the minimal schedule length in an $n = N \times N$ Mesh, and $S_T(n = N \times N)$ denote the minimal schedule length in an $n = N \times N$ Torus. Then $S_M(n = N \times N)$ satisfies:*

$$S_M(n) \leq S_T(4n) \tag{15}$$

Proof: From Lemma 1, we know the schedule length $S_T(n)$ of TNS in an $N \times N$ torus. It is easy to see that $S_T(n = N \times N)$ grows as $O(N^3)$, i.e. increases by a factor of 8, for doubling N. Moreover, we showed that we can schedule an $N \times N$ mesh using the TNS algorithm applied in a $2N \times 2N$ torus. The length of this schedule is $S_T(4n = 2N \times 2N)$. Therefore, the upper bound of $S_M(n)$ is $S_T(4n)$. ∎

The results of Theorems 6 and 7 provide an asymptotic ratio of at most 4 between the upper-bound and the lower-bound, because

$$\frac{n\sqrt{n}}{4} \leq S_M(n = N \times N) \leq S_T(4n = 2N \times 2N) = \frac{8n\sqrt{n} + 4\sqrt{n}}{8}.$$

Therefore, we are able to determine the growth rate of the periodic schedule length within a $4 + \frac{2}{n}$ - constant approximation.

VII. Greedy Scheduling Algorithms for General Periodic Traffic Pattern

The generalized scheduling problem of non-uniform traffic is NP-hard [47]. Therefore, we want to design an efficient heuristic algorithm to schedule an arbitrary traffic pattern in a general NoC topology with a general traffic demand pattern.

Algorithm Overview: Next, we introduce the *Latency-based Scheduling Algorithm*. The input of the algorithm is a periodic flow requirement that is given together with a predefined routing. The algorithm outputs a schedule that guarantees periodic service of the traffic without any collisions.

Before defining the algorithm formally, let's first provide some intuition. The algorithm runs offline. It relies on a centralized scheduler that uses given traffic demands and routes to provide a schedule. Its goal is to make the schedule period as small as possible. Intuitively, the packets that are hardest to schedule are those that take the most delay in their transmission, since they occupy more slots and therefore can experience more conflicts. Therefore, to minimize conflicts, in the latency-based algorithm, *we first schedule the packets with the longest latencies*.

Note that in the example of Figure 2, the latency-based scheduling will always produce the optimal schedule that is shown in Table II, while a random greedy scheduling can produce either of the two schedules. We will show in simulations below that latency-based heuristics also often work better in more complex cases as well.

Algorithm Description: Let's now formally introduce the latency-based scheduling algorithm. We first consider the given list of traffic demands, and sort all traffic demands by latencies in a decreasing order. Specifically, denote by $l(d_i)$ the *latency* of a flow demand i. It is defined as the length of its predefined route between the source and the destination nodes. Assuming for n flow demands that $l(d_1) \geq l(d_2) \geq \ldots \geq l(d_n)$, then the ordered set is $(d_1, d_2, \ldots, d_n)$.

The scheduler considers an empty slot table consisting of links on one axis and time slots on the second axis, for the switching schedule [31] (as illustrated in Tables I and II), and successively attempts to fill it. To do so, the algorithm iterates over the set of unallocated flows, and each time picks a flow demand with the largest latency, which is defined by the number of hops in the flow route. It then attempts to schedule the flow demand by placing it in the earliest available time-slots. Specifically, given the set of flow demands ordered by the latency, from the largest to the smallest, $(d_1, d_2, \ldots, d_n)$, the scheduler picks demand d_1 and removes it from the set. For all of the node routers on the flow route, it allocates the earliest available time-slots. The injection time of the flow is the allocated time-slot in its first node router on the route (as also described in [23]). When the flow demand set is empty, all the flows are allocated. The period length is set as the latest allocated time-slot on all the routers. For example, the period lengths of the schedules in Tables I and II are 3 and 2, respectively.

VIII. Simulations

A. Baseline Random-Greedy Algorithm

We first define the *random-greedy* algorithm as a baseline in our simulations. Our goal is to approximate the typical algorithms that consider traffic pattern demands in an online fashion, even though we compute our schedule only once at the start in an offline way. To approximate online computations, we consider the demands of the traffic pattern in a random order. To do so, we run a random permutation of all the demands, and then consider them one after the other. In other words, unlike in our latency-based scheduling algorithm, we do not order flows by latency before scheduling them. We skip the ordering by latency step of our latency-based scheduling algorithm. Therefore, the scheduler iterates over a set of unallocated flows, randomly picks one and attempts to schedule it in a greedy manner, by placing it in the earliest available slots.

While the *online* random-greedy algorithm obtains a single schedule, we can obtain a better result in our *offline* setting. To do so, we run the algorithm many times using different random permutations of the demand order, and obtain the schedule with the lowest period length out of all runs. In our simulations graphs we denote the best simulation run by *random-greedy offline*. In the figures 4 we show the distribution of the performance of the different runs of the algorithm.

B. Period Length with Complete Exchange Traffic

We simulate three NoC architectures: Ring, Torus and Mesh. We assume a complete exchange traffic pattern with a unit-sized packet to send in each period from each source to each destination. The simulations are written in Matlab. Fig. 4 plots the distribution of the period length S in each of the three NoC topologies, given several algorithms. First, Fig. 4(a) plots the period length distribution in an N-Ring topology with $N = 16$. It compares the performances of the *random-greedy* and *latency-based* greedy algorithms, described in Section VII, together with the optimal *TNS* algorithm. We remind that *random-greedy* is meant to approximate the performance of an online algorithm that would service traffic requests in their random order of arrival. On the other hand, the other algorithms attempt to benefit from the fact that the entire traffic demand pattern is known in advance. To obtain an approximate practical distribution, each of the greedy algorithms is run 100 times.

We can see that TNS manages to schedule all packets within 33 slots; while the random-greedy algorithm needs between 39 and 47 slots with an average of 41.93, and the latency-based algorithm improves upon the random-greedy algorithm by reaching between 35 and 42 slots with an average of 37.94. Next, Fig. 4(b) plots the period length distribution in an $N \times N$ Torus topology with $N = 8$, i.e. $N^2 = 64$ nodes. Again, we can see that TNS, the optimal algorithm, performs significantly better than the greedy algorithms, with a meaningful difference with even the best greedy result out of 100 runs. Last, Fig. 4(c) plots the period length distribution in an $N \times N$ Mesh topology with $N = 8$, i.e. $N^2 = 64$ nodes. This time, we do not know the optimal algorithm. Yet we see that the distributions of the *random-greedy* and the *latency-based* greedy algorithms are disjoint after 100 runs, indicating that latency-based significantly outperforms random-greedy in this topology.

C. Scaling

We now want to study how scaling N impacts the results. We define *speedup* as the performance ratio of a specific algorithm versus the average performance of the basic random-greedy algorithm. The higher the speedup, the better the performance gain. Fig. 5 shows the speedup gained by several algorithms described versus the *average* performance of the random-greedy algorithm achieved in our simulations. The three sub-figures illustrate the three different NoC topologies: the Ring with N nodes, the $N \times N$ Torus, and the $N \times N$ Mesh. In all the sub-figures, N varies from 2 to 10. For each N we run 100 iterations of *random greedy* and *latency-based* algorithms.

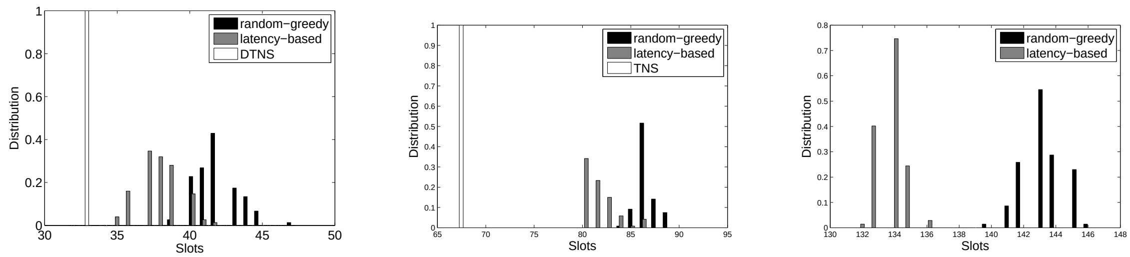
In all sub-figures, we show the best speedup result for the random-greedy algorithm (when compared to its average result). We also only show the best result for the latency-based algorithm, since it is run in offline mode, and the developer can run several iterations in order to use the best result. For the Ring and Torus we also compare with the optimal TNS algorithm. In all topologies, we can see that the variances tend to decrease as N is scaled. We can also notice successive improvements when going *from online to offline mode*, by considering the speedup of the best random-greedy run vs. its average run. Next, we can see a further improvement given the best latency-based run compared to the best random-greedy run. Finally, the optimal algorithm clearly outperforms the greedy algorithms in most cases.

D. Non-uniform Traffic

Next, we compare the algorithms using non-uniform traffic. We rely on the *permutation* synthetic pattern. The *permutation* pattern is a random traffic pattern where each node sends exactly one packet to exactly one random node, and receives exactly one packet from exactly one random node. Fig. 6 shows the distribution of the simulation results with the random-greedy and latency-based algorithms on the 8×8 Torus and 8×8 Mesh topologies for the *permutation* traffic pattern. It is generated using several random simulation runs. In all cases the latency-based algorithm achieves a better distribution.

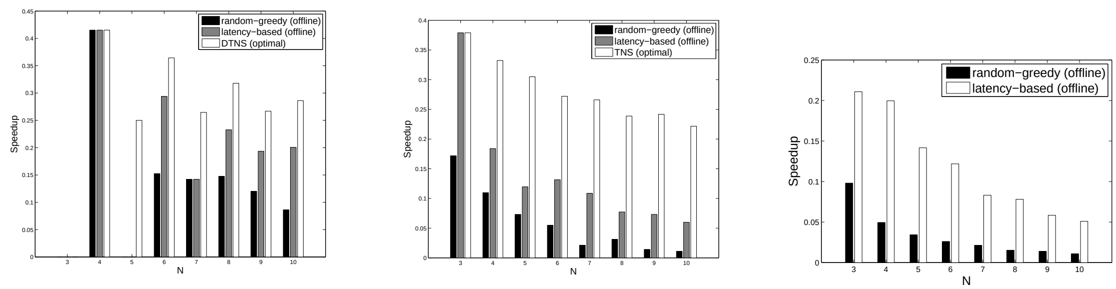
E. Analytical Evaluation

We now want to study the tradeoff between the additional capacity needed in the Torus NoC topology



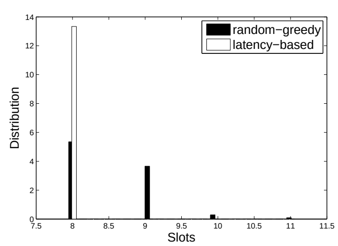
(a) Ring (b) Torus (c) Mesh

Fig. 4. Period length distribution for different topologies (smaller is better). The latency-based algorithm clearly outperforms the baseline random-greedy algorithm in all topologies. In addition, none of these algorithms can yield the same performance as our optimal DTNS and TNS algorithms in Ring and Torus.

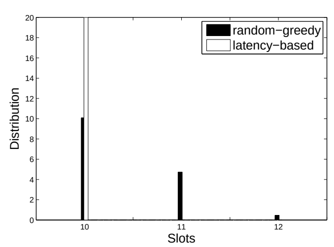


(a) Ring (b) Torus (c) Mesh

Fig. 5. Speedup for different topologies when compared to the *average* random-greedy online performance. The offline random-greedy algorithm, defined as its best run out of 100, improves upon the average online performance. The offline latency-based algorithm further improves the performance. The optimal algorithms in Ring and Torus provide a speedup above 20% in all cases.



(a) Torus



(b) Mesh

Fig. 6. Period length distribution with the *permutation* traffic pattern. In all cases the latency-based algorithm achieves a better distribution.

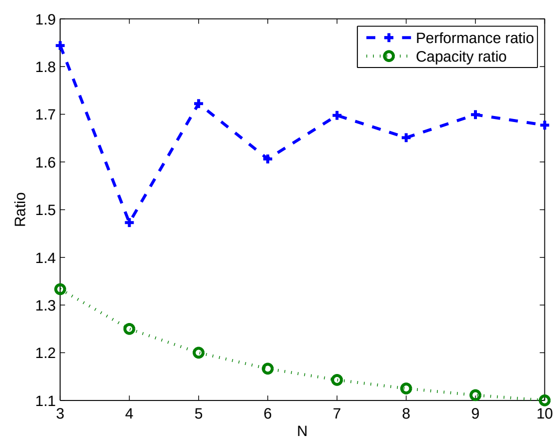


Fig. 7. Comparison of the performance ratio and the capacity ratio of the Torus and the Mesh using average online performance random-greedy algorithm.

when compared to the Mesh NoC topology, and the resulting additional performance. Figure 7 compares the performance ratio and the capacity ratio of the Torus and the Mesh NoC topologies.

- The *performance ratio* is defined as the ratio of the average period length in the random-greedy algorithm given the Torus topology, by the same parameter given the Mesh topology.
- The *capacity ratio* is defined as the ratio of the link capacities of the Torus versus the Mesh given N, i.e. $\frac{\text{Torus capacity}}{\text{Mesh capacity}} = \frac{N^2+N}{N^2}$.

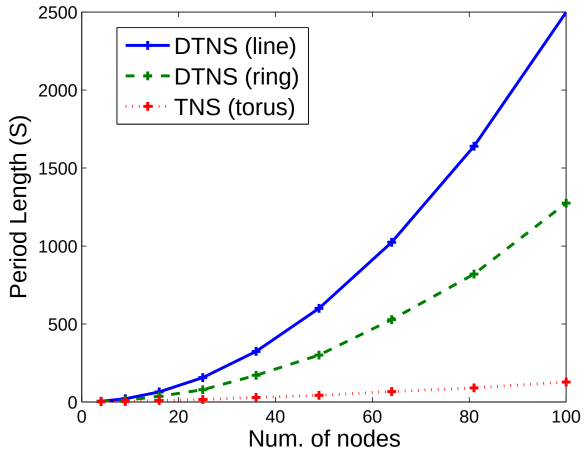


Fig. 8. Period length as function of number of nodes of the optimal algorithms in line, ring and torus topologies.

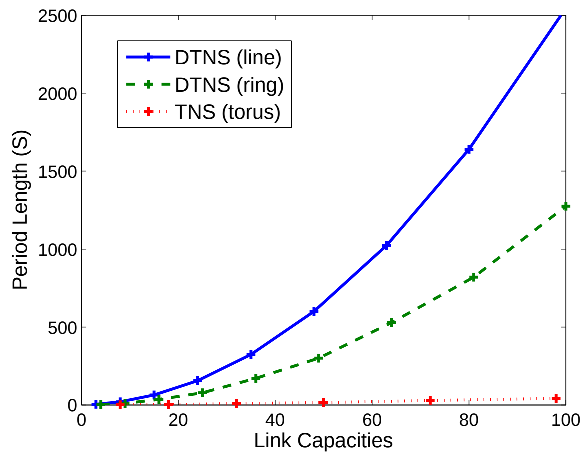


Fig. 9. Period length as function of the total link capacities of the optimal algorithms in line, ring and torus topologies. The capacity of a single link between two adjacent nodes is equal to 1.

We can see that as we scale the topologies, the performance ratio seems to stabilize around 1.68, while the capacity ratio converges to 1. At first look, this might indicate that the torus topology is more adapted. However, bear in mind that torus links might need to be longer, and therefore their cost might be higher when compared to mesh links than indicated in this simplistic model. On a more theoretical level, notice that the ratio is indeed always between 1 and 8, as proved in Theorem 7.

Next, we compare the length of the optimal schedule of DTNS and TNS algorithms under the line, ring and torus topologies as a function of the number of nodes (Figure 8) and a function of the total link capacities (Figure 9). The graphs were obtained using Properties 1 and 2 and Lemma 1. Note that the results of existence of overlapping periods is negligible with a large number of nodes (and links) and, therefore, we did not refer to it

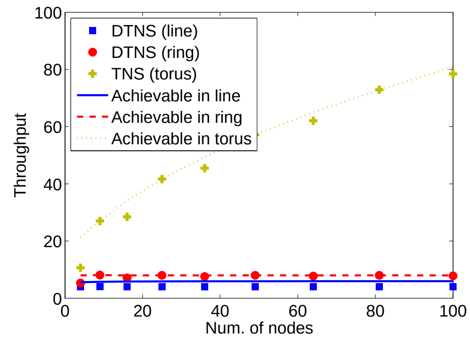


Fig. 10. Comparison of throughput of DTNS and TNS algorithms vs. the achievable capacity as function of the number of nodes.

in the plots. The results show the gained speedup of torus over degree-two topologies, and the gained speedup of ring over line.

Next, we compare the throughput of the DTNS algorithm in the line and the ring, and the TNS algorithm in the torus, with the achievable capacity in the networks under varying number of nodes. Figure 10 shows that the DTNS in a line topology is not capable of achieving full capacity, because the links in the middle of the line transmit more packets than the links closer to the edges of the line. However, DTNS in ring topology achieves full capacity under all number of nodes, as proved in Theorem 4. Finally, TNS in a torus topology achieves full capacity with odd N, and almost achieves capacity with even N. Note that it is optimal in the sense that no other schedule can achieve a better throughput in a torus topology, as proved in Theorem 5. Finally, also note that given a number of nodes, it is the torus topology that scales best, since it grows as $\Theta(\sqrt{n})$.

IX. Conclusion

In this paper, we provided several periodic scheduling algorithms for bufferless NoCs, that are designed to meet the hard communication deadlines of real-time applications, applications based on the BSP programming model, and network congestion-control applications. In particular, we introduced DTNS and TNS scheduling algorithms that were proved to be optimal for complete exchange traffic on degree-two and torus networks. We also provided an application of the TNS algorithm on mesh NoCs, and showed that it achieves a constant bounded schedule length compared to the optimal scheduling. Then, we provided a Latency-based scheduling algorithm for general periodic traffic pattern. We showed that latency-based scheduling algorithm is more efficient than random greedy scheduling on such NoC topologies as rings, tori and mesh.

In future work, we would like to investigate the following open problems: proving that the collision-free scheduling problem for periodic arbitrary traffic is NP-hard, providing improvements to our Latency-based algorithm, and to apply our results to other collision-free networks, like optical networks, which consist of arbitrary topologies.

Acknowledgments

The authors would like to thank Mark Silberstein, Ori Rottenstreich and Tsahee Zidenberg for their helpful comments.

This work was partly supported by European Research Council Starting Grant No. 210389, by the Hasso Plattner Institute Research School and by the Intel ICRI-CI Center. Erez Kantor is supported by an Eshkol Fellowship from the Ministry of Science and Technology, Israel.

References

[1] A. L. Shimpi, "Intel's Sandy Bridge architecture exposed," www.anandtech.com/show/3922/intels-sandy-bridge-architecture-exposed/4.

[2] T. Krishna *et al.*, "NoC with near-ideal express virtual channels using global-line communication," *HotI*, Aug. 2008.

[3] D. Vantrease *et al.*, "Corona: System implications of emerging nanophotonic technology," *SIGARCH Comput. Archit. News*, vol. 36, pp. 153–164, June 2008.

[4] J. Liu *et al.*, "ATAC: A manycore processor with on-chip optical network," MIT Dspace, Tech. Rep., 2009.

[5] S. Le Beux *et al.*, "Optical ring network-on-chip (ORNoC): Architecture and design methodology," *DATE*, 2011.

[6] T. Chen, R. Raghavan, J. Dale, and E. Iwata, "Cell broadband engine architecture and its first implementation," www.ibm.com/developerworks/power/library/pa-cellperf/.

[7] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, and M. Pedram, "An empirical investigation of mesh and torus NoC topologies under different routing algorithms and traffic models," *DSD*, 2007.

[8] A. Guerre, N. Ventroux, R. David, and A. Merigot, "Hierarchical network-on-chip for embedded many-core architectures," *NOCS*, May 2010.

[9] X. Chen and L.-S. Peh, "Leakage power modeling and optimization in interconnection networks," *ISLPED*, 2003.

[10] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: a fast and accurate NoC power and area model for early-stage design space exploration," *DATE*, 2009.

[11] A. Kodi, A. Sarathy, and A. Louri, "Design of adaptive communication channel buffers for low-power area-efficient network-on-chip architecture," *ANCS*, 2007.

[12] Y. Hoskote *et al.*, "A 5-ghz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, Sep. 2007.

[13] D. W. Walker and J. J. Dongarra, "MPI: A standard message passing interface," *Supercomputer*, vol. 12, 1996.

[14] R. Thakur and A. Choudhary, "All-to-all communication on meshes with wormhole routing," in *Parallel Processing Symposium*, Apr. 1994.

[15] Y.-C. Tseng and S. Gupta, "All-to-all personalized communication in a wormhole-routed torus," *Parallel and Distributed Systems*, vol. 7, no. 5, May 1996.

[16] F. Petrini, "Total-exchange on wormhole k-ary n-cubes with adaptive routing," in *IPPS/SPDP*, 1998.

[17] Y.-J. Suh and S. Valamanchili, "All to-all communication with minimum start-up costs in 2d/3d tori and meshes," *Parallel and Distributed Systems*, vol. 9, no. 5, pp. 442 –458, May 1998.

[18] Y.-J. Suh and K. Shin, "Efficient all-to-all personalized exchange in multidimensional torus networks," in *Parallel Processing*, 1998.

[19] Y.-J. Suh and S. Yalamanchili, "Configurable algorithms for complete exchange in 2d meshes," *Parallel and Distributed Systems*, 2000.

[20] D. Vainbrand and R. Ginosar, "Network-on-chip architectures for neural networks," *NOCS*, 2010.

[21] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, 1990.

[22] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta, "Globally adaptive load-balanced routing on tori," *IEEE Comput. Archit. Lett.*, vol. 3, January 2004.

[23] A. Hansson, K. Goossens, and A. Radulescu, "A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic," *VLSI Design*, 2007.

[24] R. Stefan and K. Goossens, "Multi-path routing in time-division-multiplexed networks on chip," in *VLSI-SoC*, Oct. 2009.

[25] O. Moreira, J. J.-D. Mol, and M. Bekooij, "Online resource management in a multiprocessor with a network-on-chip," *SAC*, 2007.

[26] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," *ISCA*, 2009.

[27] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis, "Evaluating bufferless flow control for on-chip networks," *NOCS*, 2010.

[28] C. Gomez, M. E. Gomez, P. Lopez, and J. Duato, "BPS: A bufferless switching technique for NoCs," *Workshop on Interconn. Netw. Arch.*, 2008.

[29] M. Hayenga, N. Jerger, and M. Lipasti, "Scarab: A single cycle adaptive routing and bufferless network," in *MICRO*, 2009.

[30] G. Nychis *et al.*, "On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects," *SIGCOMM*, 2012.

[31] K. Goossens, J. Dielissen, and A. Radulescu, "Aethereal network on chip: concepts, architectures, and implementations," *Design Test of Computers, IEEE*, vol. 22, no. 5, 2005.

[32] A. Hansson, M. Coenen, and K. Goossens, "Channel trees: reducing latency by sharing time slots in time-multiplexed networks on chip," *CODES+ISSS*, 2007.

[33] A. Hansson, M. Subburaman, and K. Goossens, "Aelite: a flit-synchronous network on chip with composable and predictable services," *DATE*, 2009.

[34] R. Stefan and K. Goossens, "An improved algorithm for slot selection in the Aethereal network-on-chip," *INA-OCMC*, 2011.

[35] T. Marescaux *et al.*, "Dynamic time-slot allocation for QoS enabled networks on chip," in *Embedded Systems for Real-Time Multimedia*, 2005.

[36] J. Liu, L.-R. Zheng, and H. Tenhunen, "A guaranteed-throughput switch for network-on-chip," in *System-on-Chip*, 2003.

[37] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip," in *DATE*, 2004.

[38] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Trans. on VLSI Systems*, 2008.

[39] Z. Lu *et al.*, "Flow regulation for on-chip communication," in *DATE*, 2009.

[40] V. Dvorak and J. Jaros, "Optimizing collective communications on 2D-mesh and fat tree NoC," in *ICN*, 2010.

[41] J. Jaros, M. Ohlidal, and V. Dvorak, "Complexity of collective communications on NoCs," *Parallel Computing*, 2006.

[42] H. C. de Freitas, L. M. Schnorr, M. A. Z. Alves, and P. O. A. Navaux, "Impact of parallel workloads on NoC architecture design," *Parallel, Distributed, and Network-Based Processing*, 2010.

[43] I. Cohen, O. Rottenstreich, and I. Keslassy, "Statistical approach to networks-on-chip," *IEEE Trans. Computers*, vol. 59, no. 6, pp. 748–761, 2010.

[44] M. Modarressi, H. Sarbazi-Azad, and M. Arjomand, "A hybrid packet-circuit switched on-chip network based on sdm," *DATE*, pp. 566–569, 2009.

[45] B. Yener, Y. Ofek, and M. Yung, "Combinatorial design of congestion-free networks," *IEEE/ACM Trans. Netw.*, vol. 5, pp. 989–1000, December 1997.

[46] I. Cidon and Y. Ofek, "Metaring-a full-duplex ring with fairness and spatial reuse," *Communications, IEEE Transactions on*, vol. 41, no. 1, pp. 110 –120, Jan. 1993.

[47] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. G. Spirakis, “Direct routing: Algorithms and complexity,” *Algorithmica*, 2006.

[48] I. Keslassy, M. S. Kodialam, T. V. Lakshman, and D. Stiliadis, “Scheduling schemes for delay graphs with applications to optical packet networks,” *HPSR*, 2003.

[49] A. Ben-Dor, S. Halevi, and A. Schuster, “Potential function analysis of greedy hot-potato routing,” *Theory Comput. Syst.*, 1998.

[50] C. Busch, “Õ(congestion + dilation) hot-potato routing on leveled networks,” in *SPAA*, 2002.

[51] U. Feige and P. Raghavan, “Exact analysis of hot-potato routing (extended abstract),” in *FOCS*, 1992.

[52] M. Adler, A. L. Rosenberg, R. K. Sitaraman, and W. Unger, “Scheduling time-constrained communication in linear networks,” *Theory of Computing Systems*, 2008.

[53] J. S. Naor, A. Rosen, and G. Scalosub, “Online time-constrained scheduling in linear and ring networks,” *Journal of Discrete Algorithms*, 2010.

[54] S. Dolev, E. Kranakis, and D. Krizanc, “Baked-potato routing,” *Journal of Algorithms*, 1999.