Footprint: Regulating Routing Adaptiveness in Networks-on-Chip

Binzhang Fu
SKL Computer Architecture ICT, CAS
Huawei Technologies Co., Ltd
fubinzhang@huawei.com

John Kim
School of Computing, KAIST
Hewlett Packard Labs
jjk12@kaist.edu

ABSTRACT

Routing algorithms can improve network performance by maximizing routing adaptiveness but can be problematic in the presence of endpoint congestion. Tree-saturation is a well-known behavior caused by endpoint congestion. Adaptive routing can, however, spread the congestion and result in *thick* branches of the congestion tree – creating Head-of-Line (HoL) blocking and degrading performance. In this work, we identify how ignoring virtual channels (VCs) and their occupancy during adaptive routing results in congestion trees with *thick* branches as congestion is spread to all VCs. To address this limitation, we propose *Footprint* routing algorithm – a new adaptive routing algorithm that minimizes the size of the congestion tree, both in terms of the number of nodes in the congestion tree as well as branch thickness. Footprint achieves this by regulating adaptiveness by requiring packets to follow the path of prior packets to the same destination if the network is congested instead of forking a new path or VC. Thus, the congestion tree is dynamically kept as *slim* as possible and reduces HoL blocking or congestion spreading while maintaining high adaptivity and maximizing VC buffer utilization. We evaluate the proposed Footprint routing algorithm against other adaptive routing algorithms and our simulation results show that the network saturation throughput can be improved by up to 43% (58%) compared with the fully adaptive routing (partially adaptive routing) algorithms.

CCS CONCEPTS

• **Computer systems organization** → **Interconnection architectures**; **Interconnection architectures**;

KEYWORDS

Congestion, routing algorithm, network-on-chip

ACM Reference format:
Binzhang Fu and John Kim. 2017. Footprint: Regulating Routing Adaptiveness in Networks-on-Chip. In *Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017,* 12 pages.
https://doi.org/10.1145/3079856.3080249

ISCA '17, June 24-28, 2017, Toronto, ON, Canada

ACM ISBN 978-1-4503-4892-8/17/06…$15.00
https://doi.org/10.1145/3079856.3080249

1 INTRODUCTION

Manycore architecture has been used to fully utilize the abundant on-chip resources provided by modern VLSI technology [3]. The design of an efficient on-chip interconnection networks, i.e., Networks-on-Chip (NoCs), remains a challenge as all aspects in the design space of NoCs needs to be considered, including the network topology, the flow control, and the routing algorithm [7]. Although different high performance topologies have been recently proposed for both on-chip and off-chip interconnection networks [22, 23, 25], this paper focuses on a 2D mesh topology since it maps well to the 2D layout or "packaging" of manycore architecture and has been implemented in commercial and experimental manycore processors [1, 10, 35, 36].

Given the topology of the network, the network performance is significantly determined by the routing algorithm. Routing algorithms can be classified as either minimal or non-minimal routing algorithms. In addition, the routing algorithm can also be classified as either oblivious or adaptive routing algorithms where oblivious algorithms route packets without considering the network status while adaptive algorithms routes packet based on the state of the network. The dimension order routing (DOR) is an example of oblivious, minimal routing while different adaptive routing algorithms [4, 11–13, 18, 28, 38] have been proposed to route packets around network congestion. In this work, we explore adaptive, minimal routing for networks-on-chip and in particular, how adaptiveness can be properly regulated to minimize congestion spreading.

One of the most important metrics of adaptive routing algorithms is the routing adaptiveness, which is defined as the ratio of the number of allowed minimal paths to the total number of minimal paths [12]. For fully adaptive routing, the degree of adaptiveness is 1 while for partially adaptive routing, it is between 0 and 1. In general, the routing algorithm adaptiveness should be maximized to improve the chances of routing around congested regions in the network. However, increasing routing adaptiveness can be a double-edge sword. If the congestion happens within the network or at a network channel (i.e. ***network*** congestion), increasing routing adaptiveness helps to route packets around the network congestion. However, if the congestion occurs at the endpoint because of an oversubscribed endpoint (i.e., ***endpoint*** congestion), increasing routing adaptiveness can lead to congestion spreading [20] or may distribute the congestion across more ports and virtual channels.

Endpoint congestion is not a problem by itself but it can cause tree saturation [32] and impact other traffic. For example, other prior work (e.g., network-wide VOQ [6], RECN [9], DBBM [30], XORDET [31], etc.) have proposed different router microarchitectures to remove tree saturation, with the solutions often involving a dedicated partition of the resource (i.e., buffers) to isolate tree saturation. The goal of this work is not to necessarily remove tree

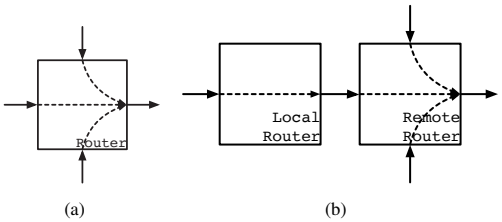

Figure 1: Examples of (a) local convergence and (b) remote convergence that create congestion in the network.

saturation but rather to minimize the impact of tree saturation while maintaining high adaptivity in the routing algorithm and provide high network throughput.

In this work, we first define the *thickness* of the branches from tree saturation as the number of VCs contributing to the same congestion tree. Thus, a *thick* branch occurs when most of the VCs contribute to the same congestion tree while a *thin* branch is defined as when the number of VCs participating in the congestion tree is minimized. In addition, we define a new, two-level routing adaptiveness where not only the port information is considered but the number of allowed VCs within each channel is also considered in the adaptive routing decision. With the goal of achieving a thin branch when congestion occurs, we propose a new routing algorithm called *Footprint* routing where packets "follow" other packets to the same destination when congestion occurs. When the network is not congested, Footprint routing prefers an output port with a larger number of free VCs [5]. However, if the network is congested, Footprint routing prefers an output port with a larger number of VCs occupied by packets with the *same* destination or the *footprint* VCs. The rationale behind *Footprint* is that if the network is congested and packets having the same destination will be blocked downstream, then it is likely that the destination is congested and the routing algorithm should constrain or regulate the adaptiveness of the packet to a limited number of VCs to reduce HoL (head-of-line) blocking. As a result, Footprint does not depend on remote congestion notification or static allocation of VCs and results in a more scalable and flexible routing algorithm compared to prior work.

In particular, the contributions of this work include the following.

(1) We propose to expand the definition of routing adaptiveness by considering the virtual channel adaptiveness or taking the number of allowed VCs into consideration.
(2) We propose a new congestion metric where we measure the number of VCs occupied by packets to the *same* destination to differentiate between network and endpoint congestion.
(3) Based on this new congestion metric, we propose a new routing algorithm, ***Footprint*** routing, to improve the network performance by regulating instead of increasing routing adaptiveness when congestion occurs in the network.

The rest of this paper is organized as follows. Section 2 discusses the HoL blocking problem caused by high routing adaptiveness and differentiates between network and endpoint congestion. We describe the proposed Footprint routing algorithm in Section 3 and evaluate Footprint in Section 4 by comparing it against both fully and partially adaptive routing algorithms. Section 5 discusses the related work and we conclude in Section 6.

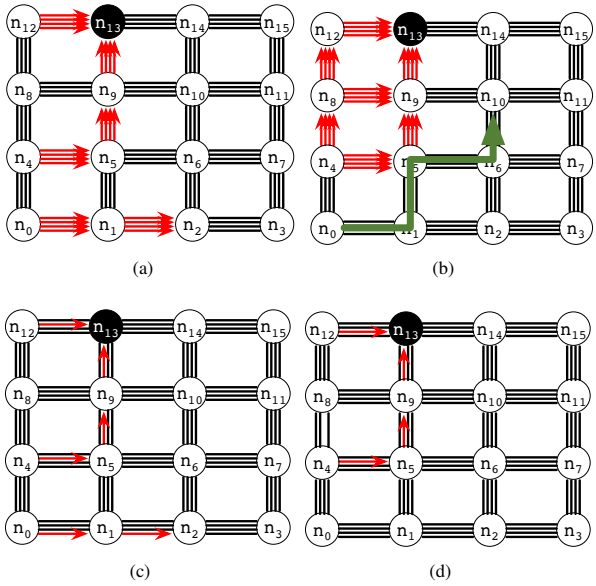

Figure 2: Impact of network and endpoint congestion with (a) DOR routing, (b) minimal fully-adaptive routing, (c) HoL-blocking-aware routing (XORDET [31]), and (d) an ideal solution. Each physical channel is assumed to have 4 VCs with the black arrow representing non-congested VC and the red arrow representing congested VC. The shaded node represent oversubscribed endpoint.

2 PROBLEM STATEMENT

For typical NoC routers, the input and output ports are homogeneous, and thus a single input port cannot saturate an output port. Therefore, to create congestion, convergence of traffic in the network needs to occur. Local convergence is shown in Figure 1(a) where three input ports send traffic to the same output port simultaneously and cause congestion at the output channel. The convergence can also be remote convergence (Figure 1(b)), where the convergence occurs at a remote router with the congestion propagating back to the local router. Since NoCs are often lossless, backpressure from the remote router results in congestion at the local router. While identifying local convergence is relatively easy, remote convergence is more difficult since it relies on backpressure and indirect information; if it occurs farther away in the network, it becomes more difficult. In this work, we rely on the *footprint* of the packets to help determine congestion and minimize the impact of congestion on network performance.

Congestion can be classified as either network congestion or endpoint congestion based on where the convergence occurs in the network [20]. If the congestion (or the convergence) initially occurs at an inter-router channel (or at a port that connects to another router), the congestion is classified as *network* congestion. However, if the router port is connected to an endpoint node and convergence occurs, then the congestion is referred to as *endpoint* congestion. The endpoint congestion can occur when an endpoint node is oversubscribed (e.g., hotspot traffic) or if the bandwidth (ejection rate) of the endpoint node is lower than the router port bandwidth. While both types of congestion can be problematic, endpoint congestion

can have a greater effect since it often leads to tree saturation [32] and creates blocking within the network.

Most prior work do not effectively address both types of congestion. To highlight these problems, an example traffic pattern that causes both types of congestion is shown in Figure 2, for a 4×4 2D mesh with 4 VCs per physical channel. Permutation traffic is considered with the following four flows:

$$\{f_1, f_2, f_3, f_4\} = \{n_0 \rightarrow n_{10}, n_1 \rightarrow n_{15}, n_4 \rightarrow n_{13}, n_{12} \rightarrow n_{13}\}$$

With DOR (dimension-order routing), f_1 and f_2 will converge on the link $(n_1 \rightarrow n_2)$ to create network congestion. Since all VCs can be used for DOR, all VCs in link $(n_1 \rightarrow n_2)$ become saturated and congestion propagates to n_0, resulting in saturating all VCs of link $(n_0 \rightarrow n_1)$ as well. In addition, f_3 and f_4 will oversubscribe n_{13} and cause endpoint congestion. The endpoint congestion also propagates backwards to both source nodes (n_4 and n_{12}), creating a congestion tree with four links and 16 VCs, as shown in Figure 2(a). Thus, DOR does not address either network or endpoint congestion.

To tolerate network congestion, fully adaptive routing algorithms can improve performance by increasing the routing adaptiveness. Since traffic can use different minimal paths, network congestion can be addressed by f_1 routing around network congestion, as shown in Figure 2(b) with a green arrow. However, fully adaptive routing does not improve endpoint congestion (i.e., n_{13}) but can actually make it *worse* since the congestion is spread to other minimal links, spreading congestion to three other links and 12 VCs – thus not only maintaining the thick branches from DOR but also creating more branches by spreading the congestion.

HoL-blocking-aware routing algorithms (e.g., XORDET [31]) can alleviate endpoint congestion by distributing traffic to different VCs, as shown in Figure 2(c). For XORDET [1], VC0 will be used for both f_3 and f_4, while VC2 is used for both f_1 and f_2 as shown in Figure 2(c). By statically assigning the VCs, XORDET creates a congestion tree with the same shape as DOR routing, but the branches are much thinner for the endpoint congestion. Thus, packets to other destinations can bypass the congested channels using other VCs. However, XORDET does not address the network congestion between f_1 and f_2. Ideally, the routing algorithm should combine the advantages of both fully adaptive routing and HoL-blocking-aware routing algorithms to create a congestion tree, as shown in Figure 2(d) – minimizing the impact of the congestion tree on the VCs and the different paths while maximizing adaptiveness. Although not shown, the network congestion still occurs but fully adaptive routing can route traffic around network congestion. In the remainder of this paper, we describe the proposed *Footprint* routing algorithm that attempts to approach this ideal goal.

3 THE FOOTPRINT ROUTING

The proposed Footprint routing algorithm is based on a fully-adaptive routing algorithm to provide high adaptivity in the routing decision. However, since the high adaptiveness can spread the congestion, we propose to regulate the adaptiveness and minimize both the number of unnecessary branches and the thickness (or the number of VCs) of the branches in the congestion tree. We first provide an overview of the two-level routing adaptiveness and then, describe the proposed Footprint routing algorithm in detail in this section.

3.1 Two-Level Routing Adaptiveness

As discussed earlier in Section 2, both the number and thickness of branches of the congestion tree can have a significant effect on the network performance when congestion occurs. However, most definition of routing adaptiveness do not consider the virtual channel (VC) adaptiveness as the routing algorithms often assume all VCs can be equally utilized. To improve the adaptiveness and how it reacts to congestion, we first define two types of adaptiveness.

Port adaptiveness (P_{adapt}) is defined between a pair of node n_i and n_j as the ratio of the number of adaptive output ports (*num_adapt_ports*) to the number of minimal ports (*num_minimal_ports*):

$$P_{adapt}(n_i, n_j) = \frac{num\_adapt\_ports}{num\_minimal\_ports} \quad (1)$$

Port adaptiveness is similar to traditional definition of routing adaptiveness, and can be used to reflect the diversity of physical paths allowed by the routing algorithm. For fully adaptive routing algorithms, $P_{adapt}(n_i, n_j) = 1$ for any nodes i and j while for partially adaptive routing algorithms, $0 < P_{adapt}(n_i, n_j) < 1$.

VC adaptiveness (VC_{adapt}) is defined for each channel-node pair (c_i, n_j) as the ratio of the number of adaptive VCs (*num_adapt_vcs*) to the total number of VCs (*num_of_vcs*) for a physical channel c_i, assuming n_j is the destination.

$$VC_{adapt}(c_i, n_j) = \frac{num\_adapt\_vcs}{num\_of\_vcs} \quad (2)$$

Traditional adaptive routing algorithms often ignore VC adaptiveness by assuming all VCs can be obliviously utilized. Thus, their VC adaptiveness is 0. For Footprint routing algorithm that we propose, the VC adaptiveness depends on how routing deadlock is addressed. For routing algorithms that exploit Duato's theory [8] to avoid routing deadlock while providing full adaptivity, the VC adaptiveness can be summarized with the following.

$$VC_{adapt}(c_i, n_j) = \begin{cases} 1 & \text{if } c_i \text{ is escape channel} \\ \frac{num\_of\_vcs-1}{num\_of\_vcs} & \text{otherwise} \end{cases}$$

Based on the definition of the two-level routing adaptiveness, we qualitatively compare different routing algorithms across both levels of adaptiveness and their impact on congestion in Table 1. Fully adaptive routing algorithms (e.g., DBAR [28]) provide high port adaptiveness and thus, helps to overcome network congestion. However, DBAR selects VCs obliviously – thus, poor VC_{adapt} and does not address endpoint congestion or HoL blocking. Similarly, partially adaptive routing (e.g., Odd-Even [4]) has similar behavior but does not necessarily provide good performance for network congestion, compared to fully-adaptive routing, because of limited adaptivity.

[1] We use XORDET as an example throughout this work. There are other similar approaches including DBBM [30]. However, these other work do not fundamentally change the results as these approaches often limit the impact of endpoint congestion but do not necessarily provide high adaptivity.

Table 1: Qualitative comparison of alternative routing algorithms ('+' indicates good, 'o' is fair, and '-' is poor).

	DBAR [28]	XORDET [31]	Odd-Even [4]	RECN [9]	CBCM [20]	This work (Footprint)
P_{adapt}	+	N/A	+	−	+	+
VC_{adapt}	−	N/A	−	−	−	+
Network Congestion	+	−	o	−	+	o
Endpoint Congestion	−	+	−	o	+	o
HoL Blocking	−	o	−	+	−	+

In comparison, blocking-aware algorithm (e.g., XORDET [31]) removes most of the head-of-line (HoL) blocking and minimizes the impact of endpoint congestion but does not necessarily overcome network congestion. [2] While XORDET is a static solution to address endpoint congestion and HoL blocking, dynamic solutions such as Regional Explicit Congestion Notification (RECN [9]) have been proposed where "set-aside queues" (SAQ) are dynamically created when congestion occurs and removed when congestion disappears; however, RECN does not address network congestion. Contention-based congestion management (CBCM [20]) dynamically throttles source nodes and routes endpoint-congested traffic to a separate VC to isolate the impact of endpoint congestion. However, detecting congestion with CBCM is complex and not applicable for an on-chip network router; in addition, CBCM does not address HoL blocking and does not provide any VC adaptiveness.

In comparison, the proposed Footprint routing algorithm provides high adaptiveness in terms of both the port and the VC adaptiveness to help address the network and endpoint congestion and the HoL blocking. However, a non-ideal behavior of Footprint is how it can sometime react to network congestion. Since Footprint is not able to necessarily differentiate between network and endpoint congestion, the routing decision might not be optimal for network congestion. When network congestion occurs, it is better to spread the traffic around different channels to provide higher bandwidth instead of taking the same path. However, Footprint will tend to take the same path *if* the other paths are congested. For example, assume a packet is trying to route through a port that has network congestion and a footprint VC exists. If an alternative port has endpoint congestion (to a different destination), the alternative port is also "congested" – however, endpoint congestion often results in the channels (or links) being underutilized and thus, if there are any idle VCs available, it might be better to route a packet towards the port with endpoint congestion and route around network congestion. However, Footprint will favor the port with footprint VC and not be able to take advantage of the port adaptiveness. Note that if a port has network congestion but the other ports have no congestion, Footprint will still distribute traffic to the other ports and exploit port adaptiveness. In addition, the impact of endpoint congestion can be avoided with traffic being isolated to footprint VCs. However, as we discuss further in Section 4.2.5, we do not limit the number of footprint VCs that can be used and thus, does not necessarily provide complete isolation between different traffic flow, unlike XORDET or CBCM. In the following subsection, we describe our proposed Footprint routing algorithm in detail.

3.2 Footprint Routing Algorithm

The idea of *Footprint* routing is inspired by Duato's theory [8] which can be summarized as "packets can be routed fully adaptive, but should wait on escape channels to avoid routing deadlock". Similarly, the core idea of *Footprint* routing is that "**packets can be routed fully adaptive, but should wait on *Footprint* channels to avoid HoL blocking**", where ***footprint* channels** are virtual channels occupied by packets to the *same* destination as the packet currently being routed. [3] The footprint channels are only considered or chosen if the network is congested – thus, if there is no congestion, all ports (and VCs) are equally considered.

The proposed *Footprint* routing algorithm is described in Algorithm 1. The *Footprint* routing has three steps, which include determining legal outputs, determining the appropriate output port, and then VC selection (and prioritization). The first step (line-5 to line-7) generates the set of legal output candidates. In this work, we focus on 2D mesh topology and utilize Duato's theory [8] to avoid routing deadlock – thus, there are at most two possible ports for routing, one in each dimension (P_x, P_y) with the escape port (P_{escape}) equal to either P_x or P_y. The VCs can be classified as either escape VC (VC_{escape}) or adaptive VC (VC_{adapt}). In addition, in the proposed Footprint routing, the VC can also be classified as either a footprint VC (VC_{fp}) or an idle VC (VC_{idle}).

In the second step (line-10 to line-20), two possible port candidates are compared against each other to determine the output port, based on the number of idle and footprint VCs. If the set of idle VCs is not empty, then the port is assumed to be not saturated and the port with more idle VCs is selected. If the decision cannot be made based on the number of idle VCs, the port with more *Footprint* VCs is selected. Otherwise, the output port is randomly selected.

In the third step (line-17 to line-39), the priority of each VC is determined based on the current state of the network. In this work, we estimate congestion in the network by comparing the number of idle VCs ($size(VC_{idle})$) with a predefined threshold (line-28). In this work, we use a threshold value of half the number of VCs per physical channel. If the network has low load and not congested, any of the adaptive VCs can be used and are requested (line-31) as waiting on footprint channels can increase packet latency. If the network is congested and then are no idle VCs, only the footprint VCs are requested if VC_{fp} is not empty; if VC_{fp} is empty and there

[2] The VC allocation techniques can be orthogonal to the routing algorithms and thus, both P_{adapt} and VC_{adapt} are not directly applicable and noted with N/A in Table 1. In our evaluation, we combine XORDET with other adaptive routing algorithms to understand the benefits of XORDET with adaptive routing algorithms.

[3] The analogy of Footprint channel is similar to the footprints found on the a snow-covered ground. There is no knowledge of what is under the snow but the "safest" way to walk through the snow-covered ground is to follow the footprint of someone who just walked through the same grounds.

Algorithm 1 *Footprint* Routing Algorithm Description.

```
1:  // size(vc_set) = number of vc's that belong to vc_set
2:  // ADD(P, v, pri) = add request for VC v at port P with
        priority pri
3:
4:  // STEP 1: Determine legal output ports and VCs
5:  (P_x, P_y, P_escape) = GetMinimalPorts(cur, dest)
6:  (VC_idle_x, VC_idle_y) = GetIdleVCs(P_x, P_y)
7:  (VC_fp_x, VC_fp_y) = GetFootprintVCs(P_x, P_y, dest)
8:
9:  // STEP 2: Determine output port direction
10: if size(VC_idle_x) > size(VC_idle_y) then
11:     route_x = true
12: else if size(VC_idle_x) < size(VC_idle_y) then
13:     route_x = false
14: else if size(VC_fp_x) > size(VC_fp_y) then
15:     route_x = true
16: else if size(VC_fp_x) < size(VC_fp_y) then
17:     route_x = false
18: else
19:     route_x = Random(1)    // randomly break ties
20: end if
21:
22: if (route_x) then
23:     P = P_x; VC_idle = VC_idle_x; VC_fp = VC_fp_x
24: else
25:     P = P_y; VC_idle = VC_idle_y; VC_fp = VC_fp_y
26: end if
27:
28: // STEP 3: Determine VC requests
29: if size(VC_idle) ≥ size(VC)/2 then
30:     // if no congestion, use all adaptive VCs
31:     Add (P, VC_adapt, Low)
32: else if size(VC_idle) == 0 then
33:     if size(VC_fp)! = 0 then
34:         Add (P, VC_fp, High)
35:     else
36:         // if no footprint VC, request all adaptive VC
37:         Add (P, VC_adapt, Low)
38:     end if
39: else
40:     Add (P, VC_idle, Highest)
41:     Add (P, VC_fp, High)
42:     Add (P, VC_busy, Low)
43: end if
44:
45: Add (P_escape, VC_escape, Lowest)
```

is no "footprint", all adaptive VCs are requested. If the network is neither near zero-load nor congested, all adaptive VCs are requested but with different priorities, as shown in line-40 to line-42. The request for the escape channel is also added with the lowest priority as well (line-45). Based on the priority of the VC requests added, the VC allocator allocates the VCs to each packet.

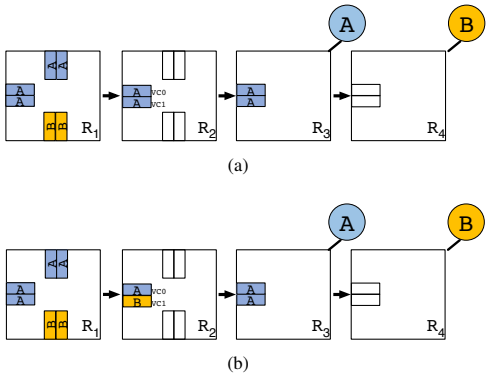

Figure 3: (a) Example of endpoint congestion and (b) how Footprint routing leverages the footprint channel to minimize blocking.

Based on Algorithm 1, *Footprint* routing maintains the high port adaptiveness, similar to other fully-adaptive routing algorithms, as well as VC adaptiveness, but higher priority is given to the *footprint* VCs over other busy (or occupied) VCs. If the network is not congested, idle VCs will be selected to maximize buffer utilization and network throughput. However, if the network is congested, packets will wait on *footprint* channels and the congestion tree will be kept "slim" while minimizing the impact of HoL blocking. In addition, by prioritizing VCs, inputs carrying packets belonging to different flows will request different VCs and thus, reduce the possible contention of VC allocation as well.

3.3 Footprint Routing Example

In this section, we provide an example of how the proposed *Footprint* routing reacts to congestion. In Figure 3(a), assume endpoint congestion for node *A* leads to accumulation of packets to node-A across routers R_1, R_2, and R_3 and assume 2 VCs per physical channel. [4] At R_1, the congestion is spread across two ports. In addition, assume packets to node-B is injected from the south input port of R_1, and the packet also needs to be routed through the east output port. This results in three packets in R_1 contending for the two VCs of the west input port of R_2. Initially, all VCs are *Footprint* VCs for packets to node-A. Thus, both packets in the north and west input ports of R_1 can request both VCs with the the actual VC granted to each packet depending on the VC allocator. Since there are no *Footprint* VCs for packets to node-B, the packet from the south port will also request both VCs. At some point, the south port will win the arbitration and without loss of generality, assume the arbiter grants $VC1$ to the south input port or the packet destined to node *B*. The key difference with Footprint is how routing (and VC allocation) is done afterwards – in the subsequent VC allocations, packets to node *A* will *always* be granted $VC0$ (both from the north and the west input ports) as it remains the footprint VC for packets to node *A* while packets to node *B* will be granted $VC1$ since $VC1$ becomes its footprint VC – thus, isolating the impact of endpoint congestion for node A to a separate VC (Figure 3(b)). Although not shown, similar allocation of VCs

[4] For simplicity, the escape VC is not shown.

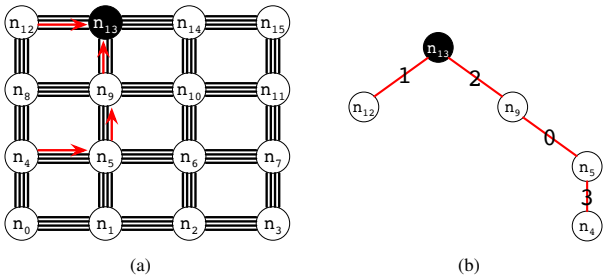

Figure 4: (a) Example of endpoint congestion in 2D mesh and (b) resulting congestion tree with the branches identified by the VC used.

will be done in R_3 for the west input port – thus, traffic to node-A will not cause any head-of-line blocking for traffic to node-B.

Once footprint channel is established for an endpoint congestion, *footprint* channels wait on each other one by one, and essentially creates a set of buffers or queues to isolate the impact of the endpoint congestion and minimize the impact of HoL blocking. The result behavior is similar to set-aside queues (SAQs) proposed as part of RECN [9] congestion management or using a separate dedicated VC [20] – however, footprint channels are effectively *virtual* SAQs since they are not physically dedicated queues but formed through the footprints and dynamically created and "destroyed". Example of an endpoint congestion with Footprint routing is shown in Figure 4(a) and the corresponding congestion tree is shown in Figure 4(b). The VCs occupied by packets destined to the n_{13} dynamically form a separate queue or a virtual SAQ – created by the footprint channels waiting on each other, instead of having a separate, dedicated buffer. As shown in Figure 4(b), different VCs can be utilize for each hop, thus the congestion tree does not consist of the same VC index, compared to other approaches (e.g., XORDET [31]). Note that Figure 4 shows the ideal result of *Footprint* and the actual shape of the congestion tree depends on the traffic pattern and the network. Thus, it is possible that the congestion tree consists of redundant branches and VCs but the possibility is significantly reduced.

3.4 Deadlock in Footprint Routing

Footprint routing is deadlock-free as it is based on Duato's theory [8] which has been shown to be deadlock-free. The only additional dependency that is introduced with Footprint routing algorithm is the dependency of allocating packets to footprint VCs. If such footprint channel dependency can cause indefinite blocking, it can possibly lead to deadlock. The footprint channel can have dependencies to two different types of downstream router virtual channels – footprint channel or non-footprint channel. If the down-stream channel is non-footprint, then escape VC can be used to guarantee deadlock freedom, similar to Duato's theory [8]. However, if the downstream channel is also footprint VC, the packets are destined to the same destination and the footprint VC can be blocked because of downstream footprint channels. If the footprint channel is continuously connected to the destination, then the packet is only dependent on the endpoint – thus, as the endpoint drains, the footprint channels will be serviced and the packets will not be blocked indefinitely.

4 EVALUATION

In this section, we compare the performance of the proposed *Footprint* routing algorithm with a fully adaptive routing algorithm (DBAR [28]), a partially adaptive routing algorithm (Odd-Even [4]), and a deterministic routing algorithm (dimension-order routing (DOR)), using both synthetic traffic patterns and application traces. In addition, a recently proposed static HoL-blocking-aware VC mapping solution (XORDET [31]) is also applied to the other routing algorithms to reduce the impact of head-of-line (HoL) blocking. The scalability of routing algorithms as the network size increases and the impact of the number of VCs on the routing algorithm performance is also evaluated. The implementation cost of *Footprint* routing is also discussed in this section.

4.1 Methodology

Cycle-accurate interconnection network simulator (BookSim 2.0 [16]) is used to compare the proposed Footprint routing algorithm against alternative routing algorithms. The detailed configurations are listed in Table 2 and the baseline network topology is 8×8 2D mesh. To study the scalability of the proposed routing algorithm, 4×4 and 16×16 2D meshes are also evaluated. The different routing algorithms are configured to maximize their performance. For DBAR routing [28], the threshold to predict congestion is the half of the number of VCs per physical channel. For Odd-Even routing [4], the number of idle VCs is used to select output ports. The routing algorithms with a $+XORDET$ suffix have been extended with XORDET to reduce the impact of HoL blocking. For example, $DBAR+XORDET$ uses DBAR to select the output port but the VC selection is determined by XORDET. The number of VCs per physical channel is assumed to be 10 as the baseline [5] and different number of VCs are also evaluated as well.

To compare the routing algorithms, three representative synthetic traffic patterns, uniform random, transpose and shuffle, are used in the evaluation. In addition, a hotspot traffic, as described in Table 3, is used to evaluate the performance of routing algorithms to alleviate HoL blocking, where eight persistent flows are utilized to create endpoint congestion. The nodes that are not participating in the hotspot traffic generate "background" traffic and uniform random traffic is used for the background traffic. To highlight the ability of routing algorithms to address HoL blocking problem, the latency of packets belonging to hotspot flows are not considered but only the latency for background traffic is measured.

In addition to synthetic traffic pattern, we also evaluate the routing algorithms using PARSEC 2.0 [2] network traces generated by Netrace [15]. The traces are used as inputs to the network simulator and the network performance (i.e., latency) is measured in the evaluation. The router microarchitecture assumed is an input-queued, virtual-channel router with credit-based flow control and wormhole switching across routers. To guarantee that the router microarchitecture is not the bottleneck, an internal speedup of 2 is assumed. To understand the impact of the routing algorithm and isolate the impact of the flow control, single-flit packets are used as the baseline in the evaluation but the impact of different packet size is also evaluated.

[5] We assume 10 VCs to facilitate the implementation of VOQ_{sw} [29], but the results for VOQ_{sw} are not shown in the paper since XORDET achieves better results.

Table 2: Network simulation configuration. The default values are shown in bold.

Parameters	Values
Network Topology	4×4, **8×8**, and 16×16 2D meshes
Routing Algorithms	*Footprint*, DBAR, Odd-Even, DOR, DBAR+XORDET, Odd-Even+XORDET, DOR+XORDET
VC	2, 4, 8, **10**, 16 VCs per physical channel, VC buffer size is 4
Traffic Patterns	Uniform Random, Transpose, Shuffle, Hotspot, PARSEC traces
Packet Size	**single-flit packets**, {1,2,3,4,5,6}-flits uniformly distributed packets
Flow Control Technique	credit-based, wormhole
Allocator and Arbiter	priority-based VC allocator, Round-Robin switch arbiter
Speedup	internal_speedup = 2.0

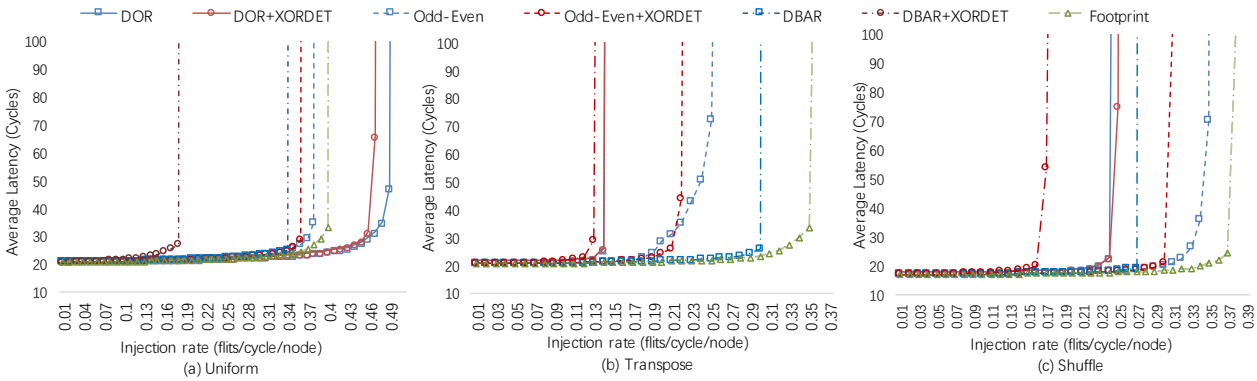

Figure 5: Latency-throughput comparison of alternative routing algorithms for (a) uniform random, (b) transpose, and (c) shuffle traffic patterns, with single-flit packet size.

Table 3: Configurations for Hotspot traffic.

Traffic	Flows
Hotspot	$f_1: n_0 \rightarrow n_{63}$, $f_2: n_{32} \rightarrow n_{63}$ $f_3: n_7 \rightarrow n_{56}$, $f_4: n_{39} \rightarrow n_{56}$ $f_5: n_{63} \rightarrow n_0$, $f_6: n_{31} \rightarrow n_0$ $f_7: n_{56} \rightarrow n_7$, $f_8: n_{24} \rightarrow n_7$

4.2 Synthetic Traffic

4.2.1 Fixed packet size. For uniform random traffic (Figure 5(a)), DOR provides the best performance since the traffic pattern itself load balances the network. By restricting the VC usage through static VC assignment from XORDET results in DOR+XORDET having slightly lower throughput. Among all of the adaptive routing algorithms evaluated, the proposed *Footprint* achieves the highest throughput for random traffic. Odd-Even achieves higher throughput than DBAR because of higher buffer utilization as routing algorithms based on Duato's theory cannot reallocate an VC unless the credit of the tail flit has been received, while Odd-Even routing does not have such restriction. However, even with higher buffer utilization, Odd-Even does not outperform *Footprint* as *Footprint* provides two-level adaptiveness and significantly reduces contention and HoL blocking to provide high performance.

As for the non-uniform synthetic traffic patterns such as transpose and shuffle traffic patterns (Figure 5(b,c)), adaptive routing algorithms outperform deterministic routing as the adaptive algorithms exploits the routing adaptiveness to distribute traffic across different paths, with *Footprint* providing the highest throughput. For these traffic patterns, statically assigning VC (i.e., XORDET) does not work well with adaptive routing algorithms as XORDET assigns a VC to a set of destinations with the limited number of VCs. However, adaptive routing further increase the interference within each set and thus, results in further performance loss. Overall, Footprint improves saturation throughput by up to 43% compared with DBAR and on average, 27% increase in performance.

4.2.2 Varied packet size. In addition to single (fixed) packet size, we also evaluate the routing algorithms where packets of different size is used in the evaluation. For this evaluation, the size of each packet is randomly distributed between single-flit packets and six-flit packets. The relative performance of fully adaptive routing algorithm is expected to improve since the utilization of buffers can be increased by the larger packet size.

For uniform random traffic (Figure 6(a)), DOR still provides the highest throughput; however, the difference between DOR and Footprint is very small. Even with larger packets and uniform random traffic, fully adaptive routing algorithm does not necessarily maximize performance (e.g., DBAR still has 15% lower throughput compared with DOR) as transient contention and HoL blocking can be increased with adaptive routing. However, by prioritizing

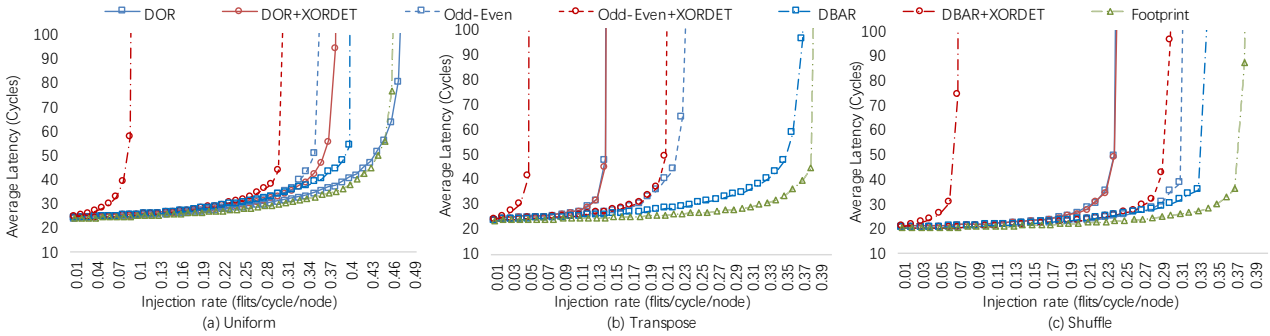

Figure 6: Latency-throughput comparison of alternative routing algorithms with variable packet size for (a) uniform random, (b) transpose, and (c) shuffle traffic patterns.

VCs, Footprint minimizes the contention and blocking to improve performance and nearly match the throughput of DOR.

For the non-uniform traffic (i.e., transpose, shuffle traffic patterns), the routing adaptiveness is exploited with Footprint to provides the highest throughput compared to other routing algorithms. By improving the buffer utilization with larger packets, DBAR significantly outperforms Odd-Even and other alternative routing algorithm. However, Footprint still outperforms DBAR across all of the traffic pattern. In addition, XORDET degrades overall performance for these traffic pattern and routing algorithms, including DOR, because of the restricted VC usage of XORDET. Since DBAR provides the highest performance among the algorithms compared against, for simplicity, the remaining results in this section will compare Footprint with only DBAR.

4.2.3 Impact of number of VCs. In this evaluation, we vary the number of VCs to understand the benefits of Footprint routing algorithm, compared with DBAR. Since Duato's theory is utilized to avoid routing deadlock, minimum number of required VCs is two. It is expected that the number of VCs can have a significant impact on overall network performance and our results show that in general, larger number of VCs provide higher throughput. However, given the same number of VCs, Footprint always outperforms DBAR with the performance benefit differing, based on the traffic pattern and the number of VCs. For uniform and shuffle traffic patterns (Figure 7(a, c)), the performance benefit of Footprint over DBAR increases with the increase in the number of VCs. Footprint improves the saturation throughput by 12.5% with 2 VCs but increases to 23.1% with 16 VCs under uniform traffic. However, for transpose traffic pattern (Figure 7(b)), the performance gain actually decreases with increase of the number of VCs – with 2 VCs, the saturation throughput is improved by 33% but with 16 VCs, the gain is reduced to 22%.

The difference in performance benefits is the result of traffic pattern characteristics. With uniform and shuffle traffic patterns, the diversity of packets within each port is higher than that of transpose traffic. Thus, chances of blocking occurring within a router is much higher and as a result, there is a larger room for *Footprint* to improve performance. However, for transpose traffic, all nodes send packets to destinations in the similar direction and thus, the diversity is low – thus, if sufficient VCs are provided, the blocking problem is alleviated and the opportunity to improve performance is smaller.

4.2.4 Different network size. We evaluate the scalability of Footprint as the network size changes in a 2D mesh by evaluating the performance on a 4×4 and a 16×16 2D mesh topology. The throughput of DBAR across the different traffic pattern is shown in Figure 8, with the throughput value normalized to that of Footprint routing for each configuration. In general, Footprint outperforms DBAR regardless of the network size and the performance gain is larger for 16×16 than 4×4 meshes since congestion can get worse in a larger network, especially for non-balanced traffic. For uniform traffic pattern, the performance gain of *Footprint* over DBAR is 11% and 13% in the 4×4 and 16×16 meshes, respectively. However, for shuffle traffic pattern, the performance improvements of Footprint are 25% and 46% respectively. For this evaluation, we do not increase the number of VCs as the network size increases but to further improve the performance of large network, more VCs can be used to increase VC adaptiveness and better address the HoL blocking problem. Alternative topologies with high-radix routers [24] can also be utilized to increase port adaptiveness and we leave the study of Footprint routing with other topologies, such as high-dimensional meshes and Dragonfly [23], as a future work.

4.2.5 Hotspot Traffic. The results presented in this section have used traffic pattern where the endpoint was not oversubscribed – i.e., the impact of endpoint congestion was limited and thus, had small impact on HoL blocking. However, one significant benefits of Footprint over other adaptive routing algorithms is the ability to limit the impact of endpoint congestion and HoL blocking. Thus, we compare the performance of hotspot traffic, described earlier in Table 3, where four nodes are selected as congested endpoints (similar to hotspot traffic that might occur with memory traffic to memory controllers).

The latency-throughput curve shown in Figure 9 is plotted differently to demonstrate the impact of head-of-line blocking from the hotspot traffic. The background traffic from the nodes not participating in the hotspot traffic is sent at a constant injection rate of 30% while the hotspot traffic injection rate is increased in the *x*-axis and the *y*-axis plots the average latency of the background traffic. The hotspot traffic will create a congestion tree and the intent of the plot is to demonstrate the impact of the congestion tree on the background traffic. The performance of DBAR is compared with Footprint with this traffic pattern in Figure 9 and the results show

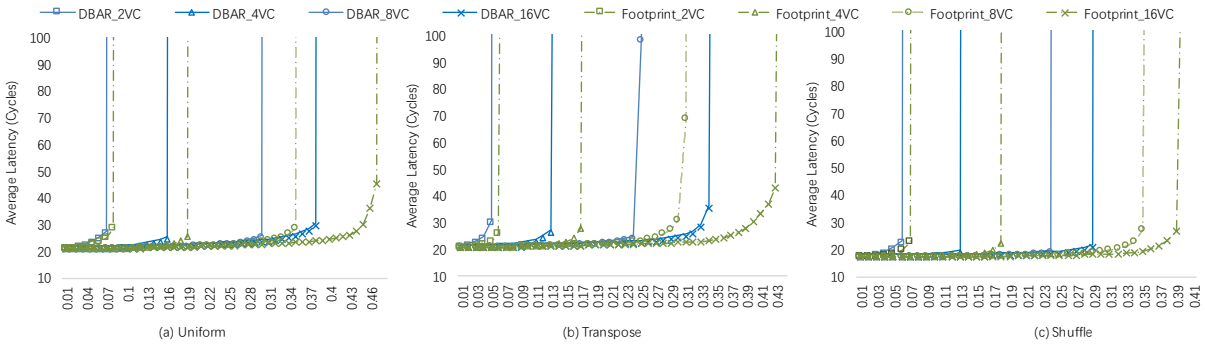

Figure 7: Latency-throughput comparison of DBAR and Footprint as the number of VCs is varied for (a) uniform random, (b) transpose, and (c) shuffle traffic patterns.

.

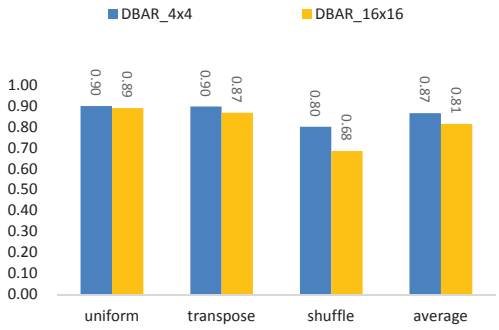

Figure 8: Throughput comparison of DBAR and Footprint for different network size. The throughput of DBAR is normalized to that of *Footprint*.

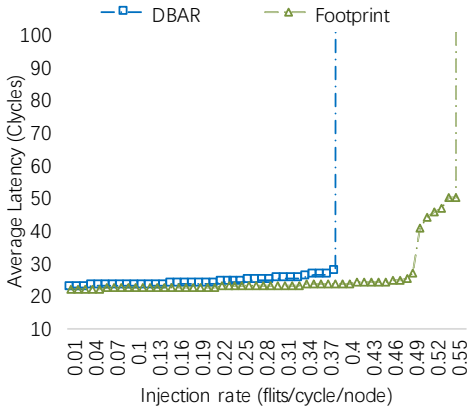

Figure 9: Routing performance comparison of Footprint and DBAR with hotspot traffic. The latency is measured for the background (uniform random) traffic that is being injected at a constant rate of 0.3 and *x*-axis represents the injection rate of the hotspot traffic nodes.

that DBAR saturates when the injection rate reaches approximately 39%, and much earlier than that of Footprint which saturates at approximately 56% offered load, resulting in over 40% improvement in throughput.

In the particular hotspot traffic used in the evaluation, two flows are injecting packets to each hotspot nodes – thus, when DBAR saturates, the total injection rate for the hotspot nodes is smaller than 100%. This suggests that the endpoints are not oversubscribed or even fully saturated but the network is heavily congested from the congestion tree – resulting in hotspot traffic causing the background traffic to saturate. However, with Footprint routing, the network is not congested until injection rate exceeds 50% or the hotspot nodes becomes saturated with two flows contributing to a single endpoint.

An ideal solution (or a solution that guarantees QoS) would result in the hotspot traffic not interfering with the background traffic. As discussed earlier, *Footprint* routing algorithm tries to alleviate the HoL-blocking problem, as a result *Footprint* could postpone but not prevent the formation of congestion tree. Based on the earlier results shown in Figure 5(a), the network does not saturate for an injection rate of 0.3. However, the HoL blocking between the background and the hotspot traffic results in the background traffic performance "collapsing" and resulted in saturation. If the background traffic was injected at a continuous rate, this would not be an issue since the hotspot traffic would prefer its footprint VC and minimize interference; however, when there is a "break" in the background traffic, the congestion tree will collapse and use up other idle VCs.

In this work, we did not limit the number of footprint VCs and by providing a mechanism to restrict the VCs can help provide better isolation between the two traffic patterns. To further solve this problem, dedicated VCs can be used either statically [30, 31] or dynamically [9, 20]. In addition, other QoS solutions can also be adopted to completely isolate the impact of hotspot traffic and achieve high network performance when the network is saturated. The techniques of adding separate VCs or other QoS solutions, including source throttling, is orthogonal to the proposed Footprint routing algorithm and can be combined. We leave that as part of future work to enable support for QoS on Footprint routing.

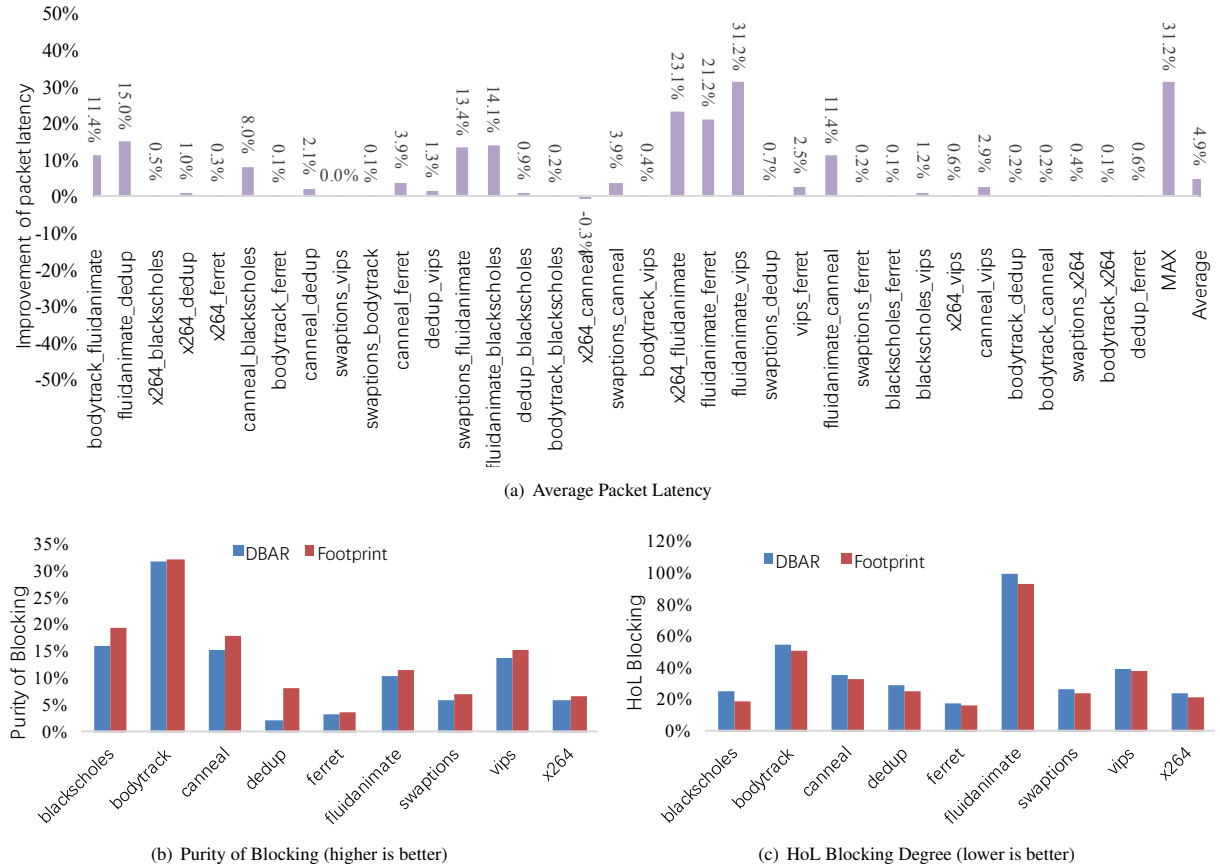

Figure 10: Comparison of Footprint and DBAR using traces from PARSEC workloads.

4.3 Application Traces

In addition to synthetic traffic patterns, we also compare DBAR and Footprint routing algorithms with network traces from PARSEC 2.0 workloads [2]. To stress the network, the two workloads are executed simultaneously and the results are shown in Figure 10(a) where the network latency different between Footprint and DBAR is shown. The results are similar to synthetic traffic patterns.

Footprint achieves better results for all cases except the combination of X264 and Canneal, where the average packet latency of *Footprint* is larger than that of DBAR by only 0.3%. For other combinations, *Footprint* outperforms DBAR by up to 31%. For workloads with small amount of network traffic, the routing algorithms has no impact. However, for other workloads that generates more network traffic, the improvement is much higher – for example, for workloads that include Fluidanimate, the average benefit is 18%. In order to benefit from Footprint, the network traffic needs to be high and HoL blocking occurs within each router.

In Figure 10(b), we track 10,000 packets for each application traces and measure the "purity of blocking" defined as the ratio of the number of *Footprint* VCs to the number of all busy VCs. Thus, to avoid HoL blocking, higher the purity is the better. Based on the analysis, Bodytrack has the highest purity (32%) and thus, it has the smallest opportunity for performance improvement while Fluidanimate's purity is approximately 10% and combined with the higher traffic load, results in higher performance improvement.

In Figure 10(c), we measure the degree of HoL blocking by multiplying the impurity ($1 - purity$) and the number of times of blocking occurred (or when VC allocation failed). The analysis shows that Fluidanimate has the highest degree of HoL blocking and results in the high improvement for Fluidanimate. Across all PARSEC workloads, *Footprint* improves the purity of blocking by up to 294% and average 44%, while reducing HoL blocking by up to 22% and average 10%.

4.4 Cost

The implementation overhead for Footprint compared with current fully adaptive routing algorithms is very small as only local router information, including the number of idle VCs and the number of *Footprint* VCs, is utilized to make the routing decisions. To track the number of idle VCs, a $\log_2(num\_of\_vcs)$ bits of register is needed for each port. where $nVCs$ is the number of VCs per physical channel. To maintain the "owner" of each VC to determine footprint VCs, a $log_2(N)$ bits register is needed for each VC, where N is the network size. For the 8×8 2D mesh with 16 VCs per physical channel, this

results in only 132 bits storage per port. Given that the size of a flit is often very wide (e.g., 128 or 256 bits), the additional storage over head is approximately equal to another flit buffer entry at each port. In addition, there is minimal impact on the critical path since the only additional complexity is determining Footprint VCs by comparing the destination of the packet with the owner of the VC.

5 RELATED WORK

Network congestion management in interconnection networks have been studied and different solutions have been proposed. Some solutions have been proposed that guarantee quality-of-service (QoS). For example, Globally Synchronized Frames (GSF [26]) guarantees that none of network resources is over subscribed within each time frame. However, a higher priority packet can be blocked by a lower priority packet in GSF and Preemptive Virtual Clock (PVC [14]) was proposed to allows the preemption of lower-priority packets and utilizes source-based retransmission to guarantee packet delivery. SurfNoC [37] maximizes the number of concurrent time frames by scheduling the network into "waves" to enable non-interference. For high priority packets, using virtual circuits based on time-division multiplexing has also been proposed [27]. Speculative reservation protocol [17] has been proposed to isolate hotspot traffic for large scale networks as well. However, the reservation-based solutions often have a tradeoff between guaranteed performance and network resource utilization. In this work, *Footprint* proposed does not limit the number of *Footprint* VCs and thus, it does not strictly isolate congested packets and can not provide QoS guarantees. However, prior work on QoS are orthogonal to this work and we expect that Footprint can be combined with previously proposed QoS solutions. In addition, an upper bound on the number of adaptive VCs can be set for *Footprint* VCs to isolate congested traffic to a fixed number of VCs and we leave that as part of future work to enhance Footprint.

To reduce the overhead of reservation-based solutions, reactive head-of-line (HoL) blocking avoidance or reduction solutions have also been proposed to remove or minimize the impact of saturation. VOQ_{net} was proposed to use dedicated VCs for each destination at each input queues [6]. Although VOQ_{net} can completely remove the HoL blocking by providing a separate set of buffer for each destination, it is not scalable as the network increases. VOQ_{sw} allocates dedicated VCs for each output port within each router [29] to reduce cost but does not completely remove HoL blocking. Destination-Based Buffer Management (DBBM) [30] solution was proposed to assign VCs to different destinations evenly and was further improved for direct networks (XORDET [31]). Since the VCs are statically assigned to different destinations, the buffer utilization will be low if the traffic is highly skewed. For example, ECN-based (explicit congestion notification) solutions [19, 33] monitor the occupancy of queues in the router and sends a congestion *alarm* to source nodes if the occupancy exceeds a predefined threshold. When the congestion notification is received, source nodes reduce their injection rate accordingly. RECN [9] exploits alternative buffer management to dynamically create dedicated set aside queues (SAQs) for congested packets to reduce HoL blocking. The footprint channels proposed in this work share some similarities to the SAQs since the footprint VCs are also formed dynamically. but there are some key differences. With SAQs, all packets which route through the congested ports use SAQs while footprint VCs are only occupied by packets going to the same destination. In addition, SAQs are separate queues that are dynamically created while *footprint* VCs are normal VCs and simplifies the buffer management in the routers. As a result, the number of SAQs are limited by the amount of hardware buffer designed while *Footprint* VCs is limited by the total number of adaptive VCs per physical channel in the network. Another important difference is that the Footprint approach is not limited to any particular routing algorithm while SAQs was proposed for source-based deterministic routing.

To overcome network congestion, adaptive routing algorithms [4, 5, 8, 11, 13, 28] have been proposed but they mainly focus on the path/port selection while *Footprint* routing in this work proposed different approach to both VC selection as well as port selection. In addition, other adaptive routing algorithms have been proposed to better address network congestion, such as global adaptive routing [34] or indirect adaptive routing [18] that includes non-minimal adaptive routing and adaptive routing on folded-Clos topology with equal hop count [21]. While these work address network congestion and improve network performance for adversarial traffic pattern that are admissible, the adaptive routing does not necessarily address endpoint congestion. The CBCM [20] was one of the first work to address the interaction between adaptive routing and congestion management to differentiate between endpoint and network congestion. However, it was proposed for off-chip, high-radix large-scale network [22, 23] and it is not necessarily applicable to low-radix on-chip networks where the hop count is much higher.

6 CONCLUSION

Most adaptive routing algorithms improve the efficiency of port selection while assuming that all adaptive VCs can be equally utilized. However, packets destined for different destination can degrade network performance when congestion occurs in the network if the virtual channel (VC) resources are not regulated. In this work, we propose *Footprint* routing algorithm that requires packets to wait on *footprint* VCs to alleviate HoL blocking when the network is congested to regulate VC usage and improve network performance. The benefits of *Footprint* comes from differentiating *footprint* VCs from other adaptive VCs when network is congested and then, prioritizing VCs accordingly. By identifying *footprint* VCs, congestion is limited to smaller number of VCs and contention of VC allocation is reduced by breaking the consensus on the "best choice" among all input ports. Through simulations, we show how *Footprint* outperforms previously proposed routing algorithms, including both fully-adaptive and partially-adaptive routing algorithms.

ACKNOWLEDGEMENTS

This work was partially supported by National Key Research and Development Program of China under Grant No. 2016YFB1000200, by National Natural Science Foundation of China (NSFC) under Grant No. 61202056, No. 61331008, and No. 61521092, by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010401. This research was also supported in part by National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (MSIP) (2015M3C4A7065647) and (2017R1A2B4011457). We thank Dr. Sheng Xu for sharing the DBAR codes.

REFERENCES

[1] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, Matthew Mattina, Chyi-Chang Miao, Carl Ramey, David Wentzlaff, Walker Anderson, Ethan Berger, Nat Fairbanks, Durlov Khan, Froilan Montenegro, Jay Stickney, and John Zook. 2008. TILE64-Processor: A 64-Core SoC with Mesh Interconnect. In *Digest of Technical Papers. IEEE International Solid-State Circuits Conference*. 88 –598. https://doi.org/10.1109/ISSCC.2008.4523070

[2] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08)*. ACM, New York, NY, USA, 72–81. https://doi.org/10.1145/1454115.1454128

[3] Doug Burger and James R. Goodman. 2004. Billion-Transistor Architectures: There and Back Again. *Computer* 37 (2004), 22–28. https://doi.org/10.1109/MC.2004.1273999

[4] Ge-Ming Chiu. 2000. The odd-even turn model for adaptive routing. *IEEE Transactions on Parallel and Distributed Systems* 11, 7 (July 2000), 729 –738. https://doi.org/10.1109/71.877831

[5] W.J. Dally and H. Aoki. 1993. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems* 4, 4 (Apr 1993), 466–475.

[6] William J. Dally, Philip Carvey, and Larry. Dennison. 1998. Architecture of the Avici Terabit Swtich/Router. In *Proceedings of Hot Interconnects Symposium*.

[7] William J. Dally and Brian Towles. 2004. *Principles and practices of interconnection networks*. Morgan Kaufmann.

[8] José Duato. 1993. A new theory of deadlock-free adaptive routing in wormhole networks. *Parallel and Distributed Systems, IEEE Transactions on* 4, 12 (Dec 1993), 1320–1331. https://doi.org/10.1109/71.250114

[9] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo. 2005. A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In *Proceedings of International Symposium on High-Performance Computer Architecture*. 108–119. https://doi.org/10.1109/HPCA.2005.1

[10] Dong-Rui Fan, Nan Yuan, Jun-Chao Zhang, Yong-Bin Zhou, Wei Lin, Feng-Long Song, Xiao-Chun Ye, He Huang, Lei Yu, Guo-Ping Long, Hao Zhang, and Lei Liu. 2009. Godson-T: An Efficient Many-Core Architecture for Parallel Program Executions. *Journal of Computer Science and Technology* 24 (2009), 1061–1073. Issue 6.

[11] Binzhang Fu, Yinhe Han, Jun Ma, Huawei Li, and Xiaowei Li. 2011. An Abacus Turn Model for Time/Space-efficient Reconfigurable Routing. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*. 259–270.

[12] Christopher J. Glass and Lionel M. Ni. 1992. The Turn Model for Adaptive Routing. In *Proceedings of International Symposium on Computer Architecture*.

[13] Paul Gratz, Boris Grot, and Stephen W. Keckler. 2008. Regional congestion awareness for load balance in networks-on-chip. In *Proceedings of the 14th International Symposium on High Performance Computer Architecture*.

[14] Boris Grot, Stephen W. Keckler, and Onur Mutlu. 2009. Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-chip. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture*. 268–279. https://doi.org/10.1145/1669112.1669149

[15] Joel Hestness and Stephen W. Keckler. May, 2011. Netrace: Dependency-Tracking Traces for Efficient Network-on-Chip Experimentation. In *Technical Report TR-10-11, The University of Texas at Austin, Department of Computer Science*.

[16] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, John Kim, and William J. Dally. 2013. A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator. In *in Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*. 86–96.

[17] Nan Jiang, Daniel U. Becker, George Michelogiannakis, and William J. Dally. 2012. Network Congestion Avoidance Through Speculative Reservation. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA '12)*. IEEE Computer Society, Washington, DC, USA, 1–12. https://doi.org/10.1109/HPCA.2012.6169047

[18] Nan Jiang, John Kim, and William J. Dally. 2009. Indirect Adaptive Routing on Large Scale Interconnection Networks. In *Proceedings of International Symposium on Computer Architecture*. ACM, 220–231. https://doi.org/10.1145/1555754.1555783

[19] S. Floyd K. Ramakrishnan and D. Black. 2001. The Addition of Explicit Congestion Notification (ECN) to IP. (September 2001). http://www.ietf.org/rfc/rfc3168.txt

[20] Gwangsun Kim, Changhyun Kim, Jiyun Jeong, Mike Parker, and John Kim. 2016. Contention-based congestion management in large-scale networks. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13. https://doi.org/10.1109/MICRO.2016.7783733

[21] John Kim, William J. Dally, and Dennis Abts. 2006. Adaptive Routing in High-radix Clos Network. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06)*. ACM, New York, NY, USA, Article 92. https://doi.org/10.1145/1188455.1188552

[22] John Kim, William J. Dally, and Dennis Abts. 2007. Flattened Butterfly: A Cost-efficient Topology for High-radix Networks. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*. ACM, New York, NY, USA, 126–137. https://doi.org/10.1145/1250662.1250679

[23] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proceedings of the International Symposium on Computer Architecture*. 77–88. https://doi.org/10.1109/ISCA.2008.19

[24] John Kim, William J. Dally, Brian Towles, and Amit K. Gupta. 2005. Microarchitecture of a high radix router. In *32nd International Symposium on Computer Architecture (ISCA'05)*. 420–431. https://doi.org/10.1109/ISCA.2005.35

[25] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, D. Frank Hsu, and Henri Casanova. 2012. A Case for Random Shortcut Topologies for HPC Interconnects. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*. IEEE Computer Society, Washington, DC, USA, 177–188. http://dl.acm.org/citation.cfm?id=2337159.2337179

[26] Jae W. Lee, Man Cheuk Ng, and Krste Asanovic. 2008. Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*. 89–100. https://doi.org/10.1109/ISCA.2008.31

[27] Z. Lu and A. Jantsch. 2008. TDM Virtual-Circuit Configuration for Network-on-Chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 16, 8 (Aug 2008), 1021–1034. https://doi.org/10.1109/TVLSI.2008.2000673

[28] Sheng Ma, Natalie Enright Jerger, and Zhiying Wang. 2011. DBAR: An efficient routing algorithm to support multiple concurrent applications in networks-on-chip. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*. 413–424.

[29] Nick McKeown, Venkat Anantharam, and Jean Walrand. 1996. Achieving 100% throughput in an input-queued switch. In *Proceedings of IEEE international Conference on Computer Communications*, Vol. 1. 296–302 vol.1. https://doi.org/10.1109/INFCOM.1996.497906

[30] Teresa Nachiondo, Jose Flich, and Jose Duato. 2010. Buffer Management Strategies to Reduce HoL Blocking. *IEEE Transactions on Parallel and Distributed Systems* 21, 6 (June 2010), 739–753. https://doi.org/10.1109/TPDS.2009.63

[31] Roberto Peñaranda, Crispin Gómez, Maria Engracia Gómez, Pedro López, and Jose Duato. 2014. HoL-Blocking Avoidance Routing Algorithms in Direct Topologies. In *Intl Conf on High Performance Computing and Communications, Intl Symp on Cyberspace Safety and Security, Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*. 11–18. https://doi.org/10.1109/HPCC.2014.9

[32] Gregory F. Pfister and V. Alan Norton. 1985. Hot spot contention and combining in multistage interconnection networks. *IEEE Trans. Comput.* C-34, 10 (Oct 1985), 943–948. https://doi.org/10.1109/TC.1985.6312198

[33] K. K. Ramakrishnan and Raj Jain. 1990. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Trans. Comput. Syst.* 8, 2 (May 1990), 158–181. https://doi.org/10.1145/78952.78955

[34] Arjun Singh. 2005. *LOAD-BALANCED ROUTING IN INTERCONNECTION NETWORKS*. Ph.D. Dissertation. Stanford University.

[35] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. 2002. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro* 22, 2 (March 2002), 25–35. https://doi.org/10.1109/MM.2002.997877

[36] Sriram R. Vangal, Gregory Ruhl Jason Howard, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Arvind Singh, Tiju Jacob, Shailendra Jain, Vasantha Erraguntla, Clark Roberts, Yatin Hoskote, Nitin Borkar, , and Shekhar Borkar. 2008. An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits* 43, 1 (2008), 29 –41.

[37] Hassan M. G. Wassel, Ying Gao, Jason K. Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, and Timothy Sherwood. 2013. SurfNoC: A Low Latency and Provably Non-interfering Approach to Secure Networks-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*. 583–594.

[38] Jongmin Won, Gwangsun Kim, John Kim, Ted Jiang, Mike Parker, and Steve Scott. 2015. Overcoming far-end congestion in large-scale networks. In *Proceedings of International Symposium on High Performance Computer Architecture (HPCA)*. 415–427. https://doi.org/10.1109/HPCA.2015.7056051