Efficient Deadlock Avoidance by Considering Stalling, Message Dependencies, and Topology

Sanya Srivastava, Fletch Rydell, Andrés Goens, Vijay Nagarajan, and Daniel J. Sorin

***Abstract*—Traditional schemes for avoiding deadlocks compose techniques for both protocol deadlocks (virtual networks) and network deadlocks (virtual channels). Recent work has shown how to use fewer virtual networks by analyzing protocol stalls instead of just considering the longest chain of causally dependent messages. We identify a shortcoming in this work, which can lead to deadlocks, and show that combining stall analysis with analyses of message dependencies and topology can avoid deadlocks while using fewer buffers than the conventional approach.**

***Index Terms*—on-chip networks, liveness**

I. Introduction

When designing a multicore processor, architects must be careful to ensure that deadlocks cannot occur in its on-chip network. Deadlocks can arise due to a bug in the coherence protocol or the network itself. Deadlocks can be classified as either protocol deadlocks or network deadlocks.

Protocol deadlocks can be caused by protocol stalling and message dependencies. Protocol stalling is when a cache or directory stalls when processing an incoming message (e.g., a cache waits to receive a Response to its Request before handling an incoming Forwarded Request). Message dependencies refer to when the reception of one message causes the sending of another message (e.g., sending Data in response to an incoming Request).

The conventional approach for avoiding protocol deadlocks—due to either cause—is using virtual networks (VNs) to segregate messages by type. With N virtual networks, each core and router has N incoming buffers per incoming link, and messages are segregated into VNs by type. The number of VNs equals the number of message types in the longest chain of dependencies. For example, a typical directory protocol has a causal chain length of 3, because a Request can cause a Forwarded Request, and a Forwarded Request can then cause a Response.

A network deadlock can occur if there is a circular dependence of buffer usage by messages. The possibility of a network deadlock depends on the topology and routing algorithm, but not on the types of messages in the protocol. As a simple example, consider an on-chip network in a ring topology with unidirectional (clockwise) routing. If every core's incoming buffer is full, and the message at the head of every incoming buffer needs to be passed to the next node in the ring, the system can deadlock. There are several techniques for avoiding network deadlocks, including virtual channels (VCs) [1], which are additional buffers used to avoid routing cycles.

These two types of deadlock—protocol and network—have fundamentally different causes and are typically handled using separate, complementary techniques. We focus on the commonly used combination of VNs and VCs, in which the cost is multiplicative. If the protocol requires N VNs and the network requires C VCs, then the processor requires $N \times C$ buffers per incoming link. The cost of these buffers is a considerable portion of the area and power of the network, and having more buffers can add to router latency.

Recently, Li et al. [2] observed that the conventional solution for avoiding protocol deadlocks is conservative, because it implicitly assumes that all messages stall. They thus use a stall-based analysis to derive a reduced number of VNs that is less than the length of the longest causal chain of messages. Their analysis considers message dependencies, except when the protocol does not stall, and that scenario can deadlock.

We show that it is possible to use the minimal number of VNs derived by Li et al. by using VCs (which are normally used for network deadlocks) to avoid message-dependency induced protocol deadlocks. Our approach takes into account knowledge of the network topology, and we do this for several commonly used topologies.

In this paper, we make the following contributions:

- We identify a protocol deadlock that can be introduced when using the VN scheme of Li et al. In doing so, we show that one must consider message dependencies even in the absence of stalls.
- For several common topologies, we show how to avoid this newly identified deadlock using VCs.
- We show that a scheme that considers stalling, message dependencies, and topology can use fewer total buffers than the traditional approach (i.e., $< N \times C$ buffers).

II. Message Dependency Protocol Deadlocks

Li et al.'s innovation for reducing the number of VNs is based on protocol stall analysis that permits dependent messages to share a VN in the absence of protocol stalling. We find this can cause a message dependency deadlock, because it creates a cyclic wait for buffers. This cyclic wait manifests as a cycle in the *channel dependency graph (CDG)* [1]. In a CDG, nodes are channels in the topology, and a directed

Received 15 June 2025; accepted 29 September 2025. This work was supported by NSF under grants CCF-200-2737 and CCF-252-5271.

Sanya Srivastava, Fletch Rydell, and Daniel J. Sorin are with Duke University (e-mail: ss1595@duke.edu, jfr27@duke.edu, sorin@ee.duke.edu).

Andrés Goens is with University of Amsterdam (e-mail: a.goens@uva.nl).

Vijay Nagarajan is with University of Utah (e-mail: vijay@cs.utah.edu).

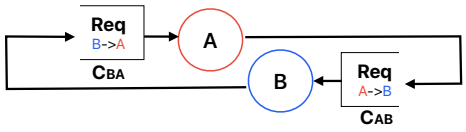

Fig. 1. Illustration of the message dependency deadlock for a 2-core processor.

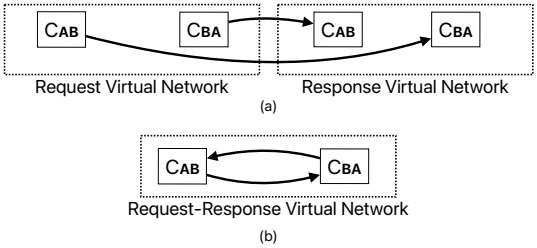

Fig. 2. (a) CDG when messages use separate VNs. (b) CDG after both Requests and Responses are placed on the same VN.

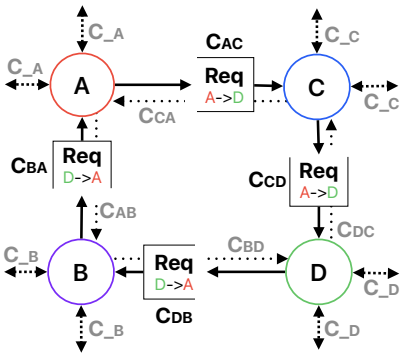

Fig. 3. Illustration of the message dependency deadlock in a portion of a mesh using XY minimal routing (i.e., messages move in X dimension before Y). A deadlock occurs because nodes A and D must process the Requests and send Responses to each other, requiring the Y-dimension channels C_{BA} and C_{CD} to request access for the X-dimension channels C_{AC} and C_{DB}. This brings back the channel dependencies that had been removed by XY routing, forming a cycle with channels C_{AC}, C_{CD}, C_{DB}, and C_{BA}.

edge from channel c_i to c_j indicates that a message queued in channel c_i can request access to channel c_j.

Consider a processor with two cores, A and B, each with a single-entry incoming buffer. The processor uses a Request/Response protocol, in which no message stalls for protocol reasons. Based on Li et al.'s work, we can use one VN and avoid protocol deadlocks (because there are no stalls in the protocol). Initially, both buffers are empty. Then, each core sends a Request to the other, as illustrated in Figure 1. The protocol allows each core to process the Request in its incoming buffer, but there is no buffer space to hold the Response it would send. A cannot send a Response to B, and vice versa, but the problem is not protocol stalling; the problem is the cycle that gets introduced in the CDG. Because Requests and Responses share a VN, Requests queued in channels C_{AB} and C_{BA} can request access to the other channel to send Responses. This creates a cycle in the CDG, as shown in Figure 2, causing a message dependency protocol deadlock.

This example makes four assumptions that are prevalent in modern processors and that we maintain in this work.

- The processing of an incoming message is atomic. A core cannot pop an incoming message if it cannot complete the work required to service that message. Thus, if an incoming message requires the core to send a message, there must be room for the outgoing message.
- Cores are unable to coordinate with each other.
- All message buffers are first-in, first-out (FIFO).
- Point-to-point ordering guarantee is preserved.

This example used a small and simple system model, but this deadlock is more broadly applicable. In Figure 3, we show an example with a 2D mesh.

III. Options for Message Dependency Deadlock

As message dependency deadlocks translate into cycles in the CDG, similar to the cycles seen in network deadlocks, we focus on using approaches used to resolve network deadlocks for mitigating message dependency deadlocks. Largely, there are four approaches to eliminating network deadlocks: flow control, routing restrictions, coordination based schemes, and virtual channels. We have chosen a combination of VCs and routing restrictions, and we explain all four here first.

Flow Control: Flow control policies have been developed that could be adapted for our use, but they either add new complexity (e.g., tracking unused buffer entries) [3] or sacrifice point-to-point ordering (e.g., Bubble Coloring [4]).

Routing Restrictions: Routing restriction algorithms, such as dimension-ordered routing (DOR) or the Turn Model [5], restrict certain turns to eliminate cycles in the CDG. Routing restrictions alone do not suffice for us, however, because message dependencies can lead to violations of the restrictions imposed by these algorithms. An example is shown in Figure 3, in which the deadlock happens despite XY routing.

Coordination Based Schemes: Coordination based schemes [6]–[10], while promising, are difficult to implement owing to their complexity, and we are unaware of any commercial implementation. Most such techniques also sacrifice some of the assumptions of our model, such as point-to-point ordering.

Virtual Channels: Virtual channels (VCs) [1] eliminate network deadlocks by breaking cyclic dependencies in routing. Because the problem we are solving differs from that in prior work—we have causally dependent messages on the same VN—we have had to perform a new analysis to determine how many VCs are necessary for our purposes.

IV. Routing to Minimize the Number of VCs

We first outline the steps used to derive our routing solutions below, and then discuss the specific routing policies obtained for rings, meshes, and tori.

- First, we construct a base CDG following the standard routing scheme of the topology, which is used when each message type has its own VN (e.g., XY routing for a 2D mesh topology), and incrementing the VC number for each propagated message. In this CDG, there are $M * V$ VCs where M is the number of messages in the dependency chain and V is the number of VCs needed for routing one message. This CDG is acyclic, but it uses the same number of buffers required if each message were scheduled on a separate VN (i.e., there is no win).

- We determine the worst-case paths, which consist of turns/movements that lead to the longest path in the base CDG from the source of the first message to the destination of the last message in the causal chain.
- We design topology-specific routing schemes that reduce VC consumption while **ensuring the worst-case routing paths are accommodated, the CDG is acyclic, and all paths are minimal**.

We present the worst-case routing paths for different topologies and a causal dependency chain of length M, with corresponding routing schemes to minimize VC usage:

- **Uni-directional ring topology**:
  Worst-case path: All M messages cross the dateline.
  Routing scheme: Every time a message crosses the dateline in VC p, it moves to VC $p+1$. Because all M messages could cross the dateline, we need $M+1$ VCs.
- **Bi-directional ring topology**:
  Worst-case path: All messages cross the dateline, and every message m' that is propagated by a message m in the chain travels in the opposite direction of m.
  Routing scheme: Message m_0 starts on VC 0. On crossing the dateline, a message m_i progresses to a VC one greater than the last VC in the same direction. If message m_i reaches its destination node n and propagates another message m_{i+1} from node n in direction d, then m_{i+1} begins on (a) VC $i+1$ if there exists *any* destination node that requires a message originating from n in direction d to cross the dateline, or (b) on VC $i+2$ otherwise.
- **K-ary N-cube topology**:
  Worst-case path: Every message crosses the dateline in every dimension, and every propagated message in the causal dependency chain turns and reverses the direction of travel in the dimension it is propagated and moves from a higher dimension to a lower dimension.
  Routing scheme: Message m_0 starts on VC 0, and the movement of a message within the ring of a given dimension is the same as for a bi-directional ring. For movement across dimensions, we follow DOR but reverse the order of dimensions for each message in the chain. For instance, if $M=2$, then the order for the first message would be $D_0, D_1, \ldots, D_{N-1}$, and order for the second message would be $D_{N-1}, D_{N-2}, \ldots, D_0$.
- **N-dimensional Mesh topology**:
  Worst-case path: All messages travel in every dimension. Every propagated message utilizes one of the turns restricted under DOR, such as reversing the direction of travel within the same dimension or moving from a higher dimension to a lower dimension, upon propagation.
  Routing scheme: Message m_0 starts on VC 0. We use DOR and reverse the dimension order for every propagated message. For a dimension, D, there are two directions: $+D$ and $-D$. Upon propagation, message m_{i+1} uses VC i (highest VC used by the predecessor message m_i) to travel in direction $-D$, and to make the turn $(+D \rightarrow -D)$, in the dimension of propagation, that is either D_0 or D_{N-1} in the worst-case path. For all other directions and dimensions, m_{i+1} uses VC $i+1$.

In all schemes, each propagated message, m_p, should start in a VC chosen from a statically determined set that depends on the VCs that m_{p-1} would use in the worst case to preserve point-to-point ordering.

Figure 4 shows the CDGs for different topologies when a causal dependency chain of length 2 ($M=2$) is placed on the same VN following the routing schemes discussed above. In these graphs, each node represents a cluster of channels that share the same dimension and/or direction associated with a particular VC number. Because the movement within this cluster will always be acyclic, cycles can only form through inter-cluster edges. Since the graphs are acyclic, the routing schemes are deadlock-free. These graphs can be extended to longer dependency chains and more dimensions, and they would remain acyclic because the routing schemes ensure no utilized node is reused. Further, the bold dashed edges show the worst-case path, showing that the number of VCs accommodates the worst case.

V. Results

For a given VN, Table I shows the number of VCs required for different topologies using our routing schemes. The baseline is $M=1$, which is the length of the dependency chain of messages scheduled on a VN prior to Li et al. [2]. In the baseline, the total number of VCs—across all VNs—is the number of VCs for $M=1$ multiplied by the number of VNs.

M	Min. VCs	M	Min. VCs
Uni-directional Ring		**Bi-directional Ring**	
1	D_0: 2	1	D_0: 2
v	D_0: $v+1$	v	D_0: $v+1$
N-dimensional mesh		**K-ary N-cube**	
1	D_0 to D_{N-1}: 1	1	D_0 to D_{N-1}: 2
v	$+D_0$: v $-D_0$: $v-\lfloor\frac{v-1}{2}\rfloor$ D_1 to D_{N-2}: v $+D_{N-1}$: v $-D_{N-1}$: $v-\lceil\frac{v-1}{2}\rceil$	v	D_0: $2v-\lfloor\frac{v-1}{2}\rfloor$ D_1 to D_{N-2}: $2v$ D_{N-1}: $2v-\lceil\frac{v-1}{2}\rceil$

TABLE I
NUMBER OF VCS REQUIRED PER VIRTUAL NETWORK FOR DIFFERENT TOPOLOGIES AND LENGTHS OF CAUSAL DEPENDENCY CHAIN.

The results show that, for all four topologies, the required number of VCs is less than M times the number of VNs. This leads to a reduction in the total number of buffers.

For K-ary N-cubes, not all dimensions have the same number of VCs. By reversing the dimension order, a propagated message begins its traversal from the dimension d in which its predecessor message would have reached its destination in the worst case. This allows the propagated message to reuse the predecessor's last VC, VC X, as its starting VC in d, needing VC $X+1$ to cross the dateline in this dimension. If each message m restarted its routing starting from the lowest dimension, then after reaching its destination node in the highest dimension on VC X, the propagated message m' would have started its routing on VC $X+1$, using VCs $X+1$ and $X+2$ in every dimension for crossing the dateline.

Thus, a propagated message first travels in the last dimension used by the predecessor message in the worst-case path, using VC X and VC $X+1$ for crossing the dateline in this

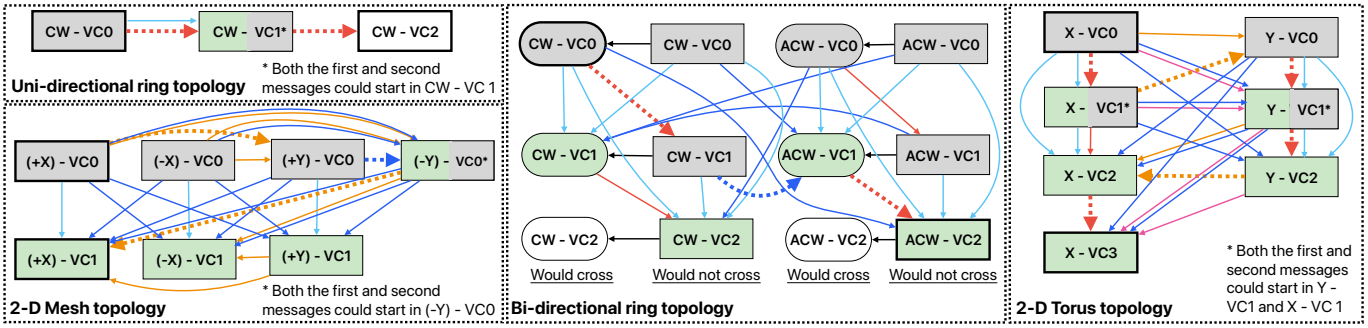

Fig. 4. CDGs for M=2. X and Y are dimensions. CW and ACW refer to clockwise and anti-clockwise. In the bi-directional ring, channels in each VC and direction are grouped by whether their associated nodes can propagate messages that need to cross the dateline. Orange edges are dimension changes, red are dateline crossings, pink are dimension changes with dateline crossings, and black are movement in the same direction. Blue edges represent message propagation: dark blue when propagation is in a different dimension or direction, and light blue otherwise. Gray-shaded and green-shaded nodes are starting VCs for the first and second messages, respectively.

dimension. Then, it moves to the other dimensions in the dimension traversal order on VC $X+1$ and uses VC $X+2$ for crossing the dateline in them. Hence, whenever the dimension traversal order is reversed, we save one VC in the dimension from which the reversal occurs. Because reversals are in the first and last dimensions, and there are $M-1$ reversals in a chain of M messages, we save $\lfloor (M-1)/2 \rfloor$ and $\lceil (M-1)/2 \rceil$ in those dimensions, respectively.

Meshes, like k-ary n-cubes, use different numbers of buffers in different dimensions. The buffer saving happens in the negative direction of the first and last dimensions, as turns from the positive to the negative direction can happen on the same VC in them. In the worst case, a message, m, after reaching its destination in dimension d, will propagate a message, m', that reverses the direction of travel in d. If this reversal is from the negative to positive direction, then m' starts on a higher numbered VC in the positive direction. However, if the turn is from the positive to negative direction, then the message can continue on the same VC number, using one fewer VC in the negative direction. Because such turns occur only in either the first or the last dimension—with half in the first dimension and the other half in the last dimension—buffer savings are only achieved in those dimensions.

Using an example of an MSI protocol that stalls only Requests, we compare the total number of buffers, with the results in Table II. In the baseline, Requests use VN 1, Fwd-Requests use VN 2, and Responses use VN 3. In our scheme, Requests use VN 1, and Fwd-Requests and Responses share VN 2. The end result is that the combination of Li et al.'s scheme for reducing the number of VNs [2] and our new routing scheme (using DOR and VCs) can lead to a modest reduction in the total number of buffers needed.

Limitations: Our routing schemes are tailored to the discussed topologies and leverage their properties to achieve a reduction in VCs. They are designed under assumptions listed in Section II and enforce constraints, such as minimal routing paths.

VI. Conclusions

We have identified a protocol deadlock that can be introduced when using Li et al's scheme for reducing the number of VNs in a system, and we have solved it using a combination of VCs and routing restrictions. The end result is deadlock freedom and a modest reduction in buffer requirements compared to the conventional approach.

Topology	Buffers: Baseline	Buffers: Reduced VNs
Rings	**6 buffers/dir** = 2 VCs/dir + 2 VCs/dir + 2 VCs/dir	**5 buffers/dir** = 2 VCs/dir + 3 VCs/dir
K-ary 3-cube	**36 buffers** = 2 VCs/dim/dir + 2 VCs/dim/dir + 2 VCs/dim/dir	**34 buffers** = 2 VCs/dim/dir + 4 VCs/dir in X & Y + 3 VCs/dir in Z
3D Mesh	**18 buffers** = 1 VC/dim/dir + 1 VC/dim/dir + 1 VC/dim/dir	**17 buffers** = 1 VC/dim/dir + 2 VCs/dir in X, Y, & +Z + 1 VC in -Z

TABLE II
TOTAL BUFFERS REQUIRED FOR EACH NODE/CORE FOR MSI PROTOCOL. BUFFERS FOR VNS 1-3 ARE SHOWN IN RED, BLUE, AND GREEN.

References

[1] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, 1987.

[2] W. Li, N. Oswald, A. Goens, V. Nagarajan, and D. J. Sorin, "Determining the minimum number of virtual networks for different coherence protocols," in *Proc. 51st Int'l Symp. on Computer Architecture*, 2024.

[3] S. Ma, Z. Wang, Z. Liu, and N. E. Jerger, "Leaving one slot empty: Flit bubble flow control for torus cache-coherent nocs," *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 763–777, 2015.

[4] R. Wang, L. Chen, and T. M. Pinkston, "Bubble coloring: Avoiding routing- and protocol-induced deadlocks with minimal virtual channel requirement," in *Proc. 27th Int'l Conference on Supercomputing*, 2013.

[5] C. Glass and L. Ni, "The turn model for adaptive routing," in *Proceedings of the International Symposium on Computer Architecture*, 1992.

[6] A. Ramrakhyani, P. V. Gratz, and T. Krishna, "Synchronized progress in interconnection networks (SPIN): A new theory for deadlock freedom," in *Proc. 45th Annual Int'l Symposium on Computer Architecture*, 2018.

[7] M. Parasar, N. Enright Jerger, P. V. Gratz, J. San Miguel, and T. Krishna, "SEEC: Stochastic escape express channel," in *SC '21: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.

[8] H. Farrokhbakht, H. Kao, K. Hasan, P. V. Gratz, T. Krishna, J. San Miguel, and N. Enright Jerger, "Pitstop: Enabling a virtual network free network-on-chip," in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture*, 2021.

[9] M. Parasar, H. Farrokhbakht, N. Enright Jerger, P. V. Gratz, T. Krishna, and J. San Miguel, "Drain: Deadlock removal for arbitrary irregular networks," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2020.

[10] M. Parasar, N. E. Jerger, P. V. Gratz, J. S. Miguel, and T. Krishna, "SWAP: Synchronized weaving of adjacent packets for network deadlock resolution," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.