# Functional Correctness for CMP Interconnects

Rawan Abdel-Khalek, Ritesh Parikh, Andrew DeOrio, Valeria Bertacco
*Department of Computer Science and Engineering, University of Michigan*
{*rawanak, parikh, awdeorio, valeria*}*@umich.edu*

***Abstract*—As transistor counts continue to scale, modern designs are transitioning towards large chip multi-processors (CMPs). In order to match the advancing performance of CMPs, on-chip interconnects are becoming increasingly complex, commonly deploying advanced network-on-chip (NoC) structures. Ensuring the correct operation of these system-level infrastructures has become increasingly problematic and, in order to avoid the potential for functional design errors manifesting into the final product, there is a need for mechanisms to safeguard communication integrity at runtime.**

**In this paper, we propose SafeNoC, an end-to-end error detection and recovery solution to ensure the functional correctness of CMP interconnects. SafeNoC augments the existing interconnect with a simple, lightweight checker network that is guaranteed to deliver messages correctly. For each data message sent over the primary NoC, a look-ahead signature is transmitted over the checker network and is used to detect errors in the corresponding data message. If a functional communication bug is detected, a novel recovery algorithm reconstructs the data that was in flight at the time of the error occurrence, ensuring that it reaches the intended destination. In our experiments, we found that SafeNoC can recover from a wide variety of errors, with almost no performance impact in the absence of errors. A lightweight solution, SafeNoC occupies a 2.41% area overhead in a 64-core CMP, 7x smaller than common retransmission-based approaches.**

## I. Introduction

Due to scaling in semiconductor technology, chip multiprocessors (CMPs) are becoming increasingly large and complex. To fully utilize the performance capabilities of these multi-core designs, interconnects have transitioned from simple buses to complex network-on-chip (NoC) architectures. In a typical NoC architecture, each processor core is connected to a router through a dedicated network interface unit. Data messages are divided into packets, which are in turn partitioned into smaller blocks called flits. Packets and flits are transmitted over the interconnect according to the network's routing protocol. To efficiently handle the communication load among the many cores on chip, NoCs are becoming increasingly complex, often implementing a wide range of topologies and providing advanced communication features, such as adaptive routing. Moreover, routers are also designed to include complex attributes such as virtual channels, pipelining, complex allocation schemes, speculation, *etc.* With such an intricate communication infrastructure, it is a challenge to ensure that the interconnect subsystem operates correctly under all execution scenarios.

Despite massive industry efforts in pre-silicon simulation, formal verification and post-silicon validation, escaped functional bugs that manifest at runtime are a reality. This is a prominent issue in processors, where design bugs are often detected after the release of the product, as can be noted in several processor errata documents [2], [13], [14], [12], [15]. As a result of the large number of interactions and the intricate communication in CMPs, errors in the communication subsystem now account for a significant portion of the reported bugs. For example, in the Core 2 Duo and Core i7, at least 10% and 13% of the design errors reported in the corresponding errata documents are associated with the communication system [13], [14], despite the CMPs having simple interconnects. As CMPs transition towards complex NoC-based interconnects, advanced router architectures, network-level interactions and concurrent communication make the interconnect highly susceptible to design errors.

Functional design errors may affect any part of the interconnect, particularly those involved in complex operations, such as virtual channel and switch allocation in routers, writing and reading from a router's input buffers, as well as the routing protocol itself. Therefore, data packets sent over the interconnect may become corrupted, misrouted, or even deadlocked. Without an appropriate runtime solution to ensure that such escaped design errors do not affect the communication correctness, these issues could lead to critical loss of data and the failure of software applications or of the entire system.

### A. Contributions

In this work, we propose SafeNoC, a novel approach to achieve runtime correctness in the communication fabric of CMPs. SafeNoC is an end-to-end solution that utilizes a novel recovery technique: upon detection of a design error, SafeNoC gathers all in-flight data, reconstructs the original packets and delivers them to their intended destination. By leveraging our novel recovery solution, we avoid the need to store redundant copies of data in-flight. To this end, we add to the baseline NoC a small and simple checker network, which operates concurrently with the primary interconnect. For each packet sent over the primary network, a corresponding look-ahead signature is sent concurrently over the checker network. Each destination node checks each data packet against the pool of look-ahead signatures that have arrived at that node. If a match is not found, a recovery process is initiated: all flits in-flight in the NoC at the time of the mismatch detection are reliably transmitted through the checker network to all destination cores. There, they are reassembled into the original packets via a software reconstruction algorithm leveraging information from the signatures available.

SafeNoC greatly enhances the functional correctness of NoCs by:

- Ensuring that all packets are correctly received at their intended destinations.
- Leveraging a novel detection and recovery approach that requires no additional storage for in-flight packets and that is implemented with small, formally verified hardware.
- Supporting any NoC topology, router architecture and routing protocol.
- Incurring low performance overhead when an error occurs.
- Maintaining a 7x smaller area footprint compared to common retransmission-based approaches and minimal performance impact in the absence of errors.

## II. Related Work

Common design-time functional verification efforts generally rely on a combination of formal verification and simulation-based validation approaches. While formal methods are effective in verifying small portions of the design, they do not scale to the size and complexity of an entire NoC, and are not effective in proving network-level properties, such as the absence of livelock and deadlock. On the other hand, simulation-based validation solutions are adequate to verify and debug the interconnect's most common behaviors. However, they cannot cover all the countless different operations and interactions of the network. As a result, critical functional design errors can often escape verification and manifest in the released hardware at runtime.

To address the limitations of design-time verification, runtime solutions have been recently proposed to ensure the correct transfer of data packets through the interconnect. Several works focus on the problem of deadlock, with some proposing deadlock-avoidance solutions by forbidding certain routes [23], and others trying to detect and recover from deadlocks. DISHA, for example, leverages timeouts for detection, and then progressively routes blocked packets through a deadlock-free dedicated link [4]. Other deadlock detection techniques, such as [19] and [18], propose more sophisticated mechanisms based on monitoring the activity at the physical channel level. In contrast, SafeNoC provides protection against several types of functional errors besides deadlock. Moreover, it uses a novel detection scheme based on sending look-ahead signatures over a checker network that is guaranteed to be functionally correct and a novel recovery approach that is based on collection of in-flight flits and reconstruction of the original data packets.

Other runtime solutions adopt more general end-to-end approaches to handle various errors in the network. A common scheme is the acknowledgment-based end-to-end error detection and recovery technique, in which a data packet is augmented with error detection codes at the source and checked for data corruption at the destination. A successful packet transfer is completed by sending back an acknowledgment message. Only after the reception of the acknowledgment, a copy of the packet stored at the source is deleted. If the acknowledgment is not received within a certain time interval, the packet can be retransmitted using the source copy. Similar approaches have also been proposed at the switch level, where upstream routers wait for acknowledgements from the downstream routers before deleting a copy of the data from their buffers[21]. Although this is a simple scheme, the additional buffer storage required for its implementation introduces significant area overhead and errors may re-occur upon retransmission. Moreover, the acknowledgment traffic degrades the overall runtime performance of the network, even in the absence of errors. In contrast, SafeNoC maintains a much lower area footprint by avoiding redundant storage of in-flight data and using instead a novel recovery process that collects flits from the network and reconstructs the original packets. It also incurs a performance impact only when a functional error manifests in the system, with a negligible overhead when there are no errors. Finally, SafeNoC does not rely on the same buggy network to recover from errors, so that it can handle a wider variety of errors than the retransmission-based approach.

Several papers have proposed solutions to overcome permanent and transient hardware faults in NoCs, such as [11], [17], [22]. In contrast, SafeNoC targets a different problem, as it aims at recovering from functional bugs that have escaped into the design.

We are not aware of any runtime solutions that deal with design errors in NoCs. However, there has been few such works for processors, [5], [20], [25]. In general, these solutions add checker hardware to verify the operation of untrusted components. SafeNoC is similar to such solutions, in the sense that it uses a simple and functionally correct checker network to verify the operation of the complex primary network.

SafeNoC's detection and recovery mechanisms rely on augmenting the original network with a small and lightweight one that operates concurrrently. The idea of using multiple overlayed networks has been proposed for various purposes. [6] and [26] use multiple networks for performance enhancement. Others, such as TILE64 [7], use separate dedicated networks, each supporting a distinct functionality of the NoC. Mostly, these networks share the same topology and are comparable to each other in complexity. To the best of our knowledge, SafeNoC is the first attempt to overlay a network with another low-cost and error-free one to ensure the functional correctness of the original interconnect.

## III. SafeNoC Architecture

### A. Overview

The SafeNoC solution relies on adding a simple and lightweight checker network that works concurrently with the original interconnect. This network is designed to be simple enough to be formally verified and guaranteed to be free of any functional errors. As a result, it provides a reliable medium through which we implement our detection and recovery processes. In the detection phase, whenever a packet is to be sent over the primary network, a signature of that packet is computed and sent through the checker network. The signature serves as a look-ahead packet and a unique identifier of the corresponding main packet, and it is used as a basis for detecting errors in the main interconnect. When a destination receives a data packet, it recomputes its signature and compares it against

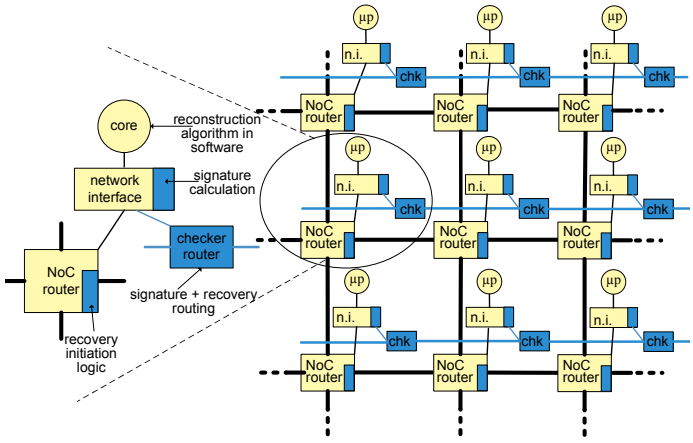


Fig. 1: **High-level overview of SafeNoC.** SafeNoC augments the original interconnect with a lightweight checker network. For every data packet sent on the primary network, a look-ahead signature is routed through the checker network. Any mismatch between a received packet's computed signature and its look-ahead signature flags an error and triggers recovery.

previously received look-ahead signatures. If a match is not found within a certain timeout period, an error is flagged and recovery is initiated. During the recovery phase, in-flight flits and packets are recovered from the network, and reliably transmitted through the checker network to all destinations. Any destination that has a mismatched signature, runs a software-based reconstruction algorithm, in which it uses the recovered flits to reconstruct the original data packets, so that they match their corresponding signature. Figure 1 shows a baseline CMP interconnect overlayed with our checker network. Both the checker router and the NoC router connect to the network interface, to which we also add two signature calculation units. Some additions to the primary NoC routers are also required for recovering in-flight flits. Once all flits have been recovered, the microprocessor cores run a software algorithm to reconstruct the original packets.

SafeNoC is not limited to a particular interconnect topology or router architecture. For our experimental evaluation, we chose a general baseline router architecture that is input-queued and uses virtual-channels and wormhole routing. We also assume the data flit width to be 64-bits and that the primary routers have built-in error-correcting code (ECC) functionalities to protect against bit-level data corruption.

### B. Checker Network

SafeNoC's checker network is designed to satisfy three main properties: i) it should incur minimal area overhead. ii) it should deploy a simple router architecture, topology and routing algorithm, so that its design can be easily formally verified. iii) Finally, it should have low latency, so that it can deliver look-ahead signatures before actual data packets arrive through the primary network. Note that these properties are not a strict requirement for the correctness of the SafeNoC solution; however additional costs in area, development time and/or performance are incurred when they are not met. A checker network that does not deliver signatures before their corresponding data packets would only introduce a performance penalty but would not prevent detection and recovery. Based on the characteristics of the primary interconnect, the checker network can be carefully designed in order to minimize the occurrence of such cases. In addition, since the checker network is designed to be simple enough to be amenable to formal verification, it can be assumed that it is free of functional bugs.

Based on the properties of the baseline primary NoC used in our evaluation, we chose a ring topology for the checker network because of its simplicity and small area overhead. In addition, since the checker network transmits look-ahead signatures of fixed size (16-bits in our case, as we discuss in Section III-C), we tailor its channel bandwidth accordingly so to achieve both efficient bandwidth utilization and area savings. To optimize the performance of the checker network, we leverage the simple, single-cycle latency, packet-switched router, based on the solution proposed in [16].

### C. Error Detection

SafeNoC's primary goal is to ensure that a packet sent over the interconnect arrives unaltered to the correct destination. SafeNoC assumes that data within a packet's flits is protected by built-in ECC, and thus it does not become corrupted during transmission. Consequently, our detection scheme considers that latent functional design errors can either manifest by affecting the delivery of entire packets, such as a deadlock, or by affecting the routing of individual flits within a packet, such as misrouting or re-ordering of flits, but not by corrupting data bits within a flit. In addition, SafeNoC does not attempt to localize the functional bug, but instead it detects when the bug compromises the functional correctness of the NoC and recovers by reconstructing the packets affected by the bugs.

For each packet sent over the primary network, SafeNoC sends a corresponding look-ahead signature packet via the checker network. The signature calculation is based on a combination of shift-XOR operations. For a signature to uniquely identify a packet, its value must depend on the flits' data values, as well as their order within the packet. As a result, every flit in the data packet must be augmented with a flit ID, which is transmitted along with the flit data on the primary network. The 64-bit data of each flit (see Section III-A) is rotated by a fixed amount that depends on the flit's position. The resulting values are XORed together into a 64-bit intermediate value. The intermediate value is divided into 4 parts that are then XORed to give the final 16-bit signature. This solution provides an effective signature mechanism, which has a very low silicon area profile. It also has no performance overhead, since the signature is calculated incrementally and concurrently with the flit's data being injected into the primary network.

We also analyzed the probability of aliasing for this 16-bit signature and found that it is extremely low. To estimate this probability, we set up a Monte-Carlo-based simulation: we randomly created 6,000 data packets and computed their signatures. We then randomly permuted each packet's flits, and

for each permutation we re-calculated the signature. If the new signature matched the original, then aliasing had occurred. For each of the 6,000 samples, we tried approximately 30 million distinct permutations, and we obtained a total probability of aliasing of $3.05 \times 10^{-5}$ with a 95% confidence interval. For interconnects with larger data widths, the signature size can be tailored accordingly so as to maintain a similarly low aliasing probability. However, the probability that an error goes undetected does not just depend on the probability of aliasing, but also on other factors such as the timing of its occurrence. As a result, the overall probability of not detecting an error is much lower than the aliasing probability.

Each destination router maintains a timeout counter for every look-ahead packet it receives. The counter is incremented at every cycle until the data packet is received and its signature is re-computed. If the new signature matches any of the look-ahead signatures, then this packet is considered to have been delivered correctly. However, if there is still no match when the counter times out, an error is flagged and recovery is initiated.

### D. Recovery

When an error is detected, the interconnect enters the recovery phase, consisting of five steps: *network drain*, *packet recovery*, another *network drain*, then *flit recovery* and *packet reconstruction*.

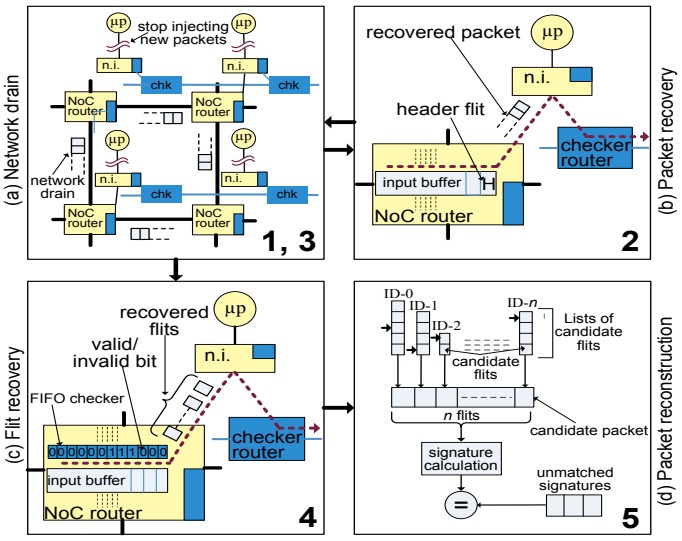


Fig. 2: **SafeNoC recovery process.** Recovery proceeds in five steps with network draining occuring twice. The last step is executed in software, while the others are implemented in hardware.

In the first step, a *network drain* phase is initiated, during which the network is forced to drain its in-flight packets for a preset amount of time. As shown in the top left corner of Figure 2, during this phase, no new packets are allowed into the network, while in-flight packets continue moving towards their destinations. If those packets are error-free, they will match their signatures and be ejected from the network. This draining stage clears the network from all in-flight traffic that was not affected by the functional bug that manifested.

The network then enters *packet recovery* and tries to recover packets that are deadlocked within the network. During this phase, primary routers remain active, but we prevent them from processing new data packets by disabling all the virtual channel allocation functionalities. If there is a deadlock in the network, then there is at least one packet in one router that is blocked waiting for allocation. To cope with this situation, we use a token-based protocol, in which a token circulates through the checker network. When a primary router notes the token in its corresponding checker router, it checks its input buffers to determine if there is such a deadlocked packet, in which case it transmits it over the checker network, as shown in the top right block of Figure 2. Since all other router functionalities are still active, the entire packet can be drained and is then transmitted over the checker network to its destination. Once the token has circulated through all primary routers, they resume their full functionality and the entire network enters the second *network drain* phase. If the previous phase had recovered packets involved in deadlocks, then the remaining packets would automatically drain from the network during this second network drain phase. Note that this situation allows us to avoid the reconstruction of a number of packets that are now being delivered through the primary network, greatly reducing the packet reconstruction computation.

```
Reconstruct (candidate_flits, signatures)
 while (!Empty(signature_buffer))
  success = false
  while(!success && !tried_all_combinations) do
   curr_candidates = GetNextCombination()
   candidate_pkt = AssemblePkt(curr_candidates)
   calc_signature = CalcSignature(candidate_pkt)
   success = MatchSig(calc_signature, signatures)
  end while
  if(success)
   RemoveCandidates(curr_candidates)
 end while
end Reconstruct
```

Fig. 3: **Packet reconstruction algorithm** The algorithm reconstructs packets by considering combinations of candidate flits. When a candidate packet's signature matches a look-ahead signature, the data packet is considered correct. Reconstruction ends when all look-ahead signatures have been matched.

In the forth step, *flit recovery*, we recover stray flits from the network. A flit is considered stray if it is stuck in a router buffer or if it has been delivered to the wrong destination. All stray flits are candidates for the final reconstruction process. The bottom left portion of Figure 2 illustrates this phase: we added a FIFO checker to every input buffer of each router, to identify valid flits. The FIFO checker has 1-bit entries and its own read and write pointers following those of the input buffer. A write to the input buffer changes a corresponding entry in the FIFO checker to a valid entry and a read invalidates it. Using the same token-based protocol as in the previous phase, a router holding the token examines its FIFO checkers for valid entries. If any exist, the corresponding flits are transmitted over the checker network to all destinations in the network.

As for stray flits at the network interface buffers, their presence in the buffers at this point of the recovery process indicates that they have not matched any signatures, thus they are also candidates for reconstruction, and they are also circulated over the checker network.

During the last phase, *packet reconstruction*, the processor cores that have flagged an error run a software algorithm to reconstruct the original packets destined to them using the flits collected in the previous steps. Candidate flits are organized in separate groups, one for each flit ID, and an index is maintained for each group to indicate which flits have already been considered. The pseudo-code of this algorithm is presented in Figure 3. For each flit ID, a candidate a candidate is chosen and added to the set of current candidates. The current candidates are then assembled into a new packet and its signature is computed. If the new signature matches any of the remaining look-ahead signatures, then this packet's reconstruction is deemed sucessful, the packet is delivered to the application and all its flits are removed from the candidate groups. If a match cannot be found, the process is repeated, generating new sets of candidates, until all possible combinations have been tried. The algorithm ends when all look-ahead signatures have been matched.

In the case of multiple functional errors occurring consecutively, SafeNoC is still able to recover successfully. After the first error is detected and recovery is initiated, any subsequent error can only manifest in the primary interconnect during one of the network drain phases. In that case, flits that failed to drain from the network are recovered during step 4, flit recovery, along with the erroneous flits resulting from the first error. Note that during SafeNoC's recovery, functional bugs can not manifest in the checker network or the recovery process since they are formally verified.

## IV. SafeNoC Implementation

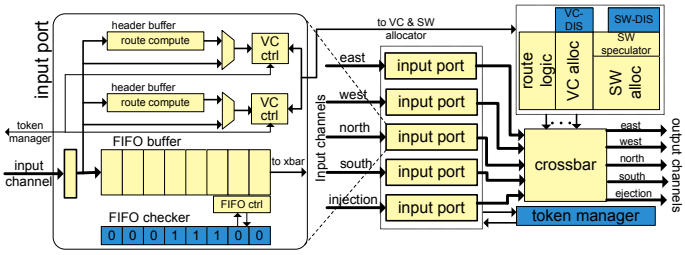


Fig. 4: **SafeNoC's hardware implementation.** VC and SW disablers, FIFO checkers, and a token-manager are added to the primary routers to implement SafeNoC's recovery.

In order to implement SafeNoC, we added a few functionalities to a baseline CMP interconnect design, categorized in three groups: to the network, to network interfaces and to the primary routers. At the network level, we add the checker network, using a ring topology with simple single-cycle latency routers, as overviewed in Section III-B. Checker routers are packet-switched, with 2-entry input buffers and use a rotary flow control, giving packets already in the network priority over packets waiting to be injected into the network, so to guarantee forward progress for packets within the checker network at each cycle. In [16], the authors show that the checker router design leads to a per-hop latency of one cycle and a deadlock-free ring network. The resulting router is sufficiently simple that it can be formally verified for correct operation, as we discuss in Section V-A, and it has very low area overhead, as shown in Section V-C. In addition, to transmit the signature packets, the checker network augments each packet with a destination address. During recovery, data flits sent over the checker network are partitioned into smaller blocks and then transmitted back-to-back, since the links of the primary network are wider than those of the checker network. SafeNoC also adds a flit ID to each data flit, slightly increasing the primary network's channel width. At each network interface, we add two *signature generation units*, one to generate signatures for packets entering the primary network and the other to verify signatures of packets reaching their destination. Signatures received through the checker network are also stored in *signature buffers* and they are compared in a *signature comparator unit* against those of incoming packets.

Figure 4 shows the router level hardware additions to the primary routers. First, a token manager is added to the routers to manage the passing of the token during the packet and flit recovery steps. In addition, a *virtual channel allocation disabler* (VC-DIS) and a *switch speculation disabler* (SW-DIS) are included to prevent routers from processing new packets during the packet and flit recovery phases. We also disable switch speculation during the entire recovery process to keep the SafeNoC operation simple and easily verifiable for correctness. Besides these modifications, the packet recovery phase is implemented with very little overhead, as it relies on the router's existing functionalities to retrieve packets from the buffers, with the exception of a FIFO checker for every input buffer to keep track of valid entries.

## V. Experimental Results

To evaluate SafeNoC, we modeled a CMP system in Verilog HDL and with a cycle-accurate C++ simulator. Using the hardware implementation, we formally verified the portion of the system involved in recovery, including the checker network, ensuring that it operates correctly. We also analyzed the area overhead of the SafeNoC solution, synthesizing the Verilog design with a 45nm target library. The impact of recovery on performance was evaluated using the C++ simulator modeling a variety of functional bugs in the baseline system. The model was simulated with two different types of workloads: directed random traffic (uniform, transpose and bit complement), as well as application benchmarks from the PARSEC suite [8].

Both the C++ and Verilog experimental setups model the same baseline system, based on the Booksim [10] simulator. The main network, an 8x8 mesh using XY routing, was augmented with a ring checker network and modified to include detection, recovery and reconstruction functionalities. The main NoC routers are based on the input-queued VC router of [10], with 5 ports, 4 pipeline stages, 2 virtual

channels, and 8 flit buffers. Data packets consist of 16 flits of 75 bits each, including ECC and flit IDs. We also integrated SimpleScalar [9] in our architectural simulation to estimate the reconstruction algorithm's execution time. The network drain time (steps 1 and 3 of recovery) was set to 2,000 cycles and the packet delivery timeout to 4,000 cycles.

### A. Correct Functionality

To guarantee correct functionality and forward progress of detection and recovery, all components involved in these processes must be formally verified. Detection is verified by ensuring that un-matched signatures initiate recovery after the timeout threshold. Recovery initiation by a single router can be trivially verified and thus we approached the formal verification of the recovery process in two steps:

**Checker network.** We must verify the functionality of the checker network to ensure that all packets are delivered unaltered to their correct destination within a bounded time. This goal was partitioned into three sub-goals: *eventual injection*, which guarantees that a packet awaiting injection will eventually enter the network; *forward progress* ensuring that packets progress on a path towards their destination; and *timely ejection* guaranteeing that packets are eventually ejected at the correct destination. Since the checker network is designed to be simple, its formal verification is not as challenging as that of the primary interconnect. The formal verification is even further simplified by the fact that the checker network transmits one flit signatures and not large data packets, which makes its router architecture and protocol inherently simpler.

**Interaction with the main network.** We must also verify that the checker network interacts correctly with the primary network to recover the data in transit. The large state space of the complex baseline network is a challenge for formal verification. We overcome it by disabling all hardware units not involved in the recovery process, such as the VC allocators and SW speculators. We first verified that the checker network could extract a complete packet from a primary router. Next, we verified that during flit recovery, all valid flits are extracted from the primary network router. We also validated the complement of the two properties above, to check that only valid data is extracted. Finally, we checked for fairness and exclusivity among the primary routers during recovery, verifying that data is salvaged from one router at a time.

All properties to be verified were expressed with System Verilog Assertions [1], embedded in a 2x2 mesh version of the CMP equipped with SafeNoC, and then formally verified with Synopsys' Magellan [24]. Since the checker network has a simple topology, router architecture and protocol, its formal verification was completed without obstacles. As for the verification of the interactions with the main network, it is sufficient for us to formally verify these properties for a smaller 2x2 interconnect. Indeed, during recovery only one router in the primary network is active at a time, eliminating complex interactions and concurrent communication in the network. Note that this aspect does not hold true during normal operation, during which the correctness of a smaller portion of network does not imply correctness of the full system.

### B. Performance

| Bug name | Bug description |
|---|---|
| dup_flit | a flit is duplicated within a packet |
| misrte_1flit | a flit is misrouted to a random destination |
| misrte_3flit | 3 flits of a packet are misrouted to a random destination |
| misrte_1pkt | a packet is misrouted to a random destination |
| misrte_2pkt | 2 packets are misrouted to random destinations |
| misrte_flit_pkt | a packet is misrouted, another packet's flits are misrouted |
| dup_pkt | a packet is duplicated |
| dup_misrte_pkt | a packet is duplicated and one copy is misrouted |
| reorder_flit | flits within packet are reordered |
| deadlock | some packets are deadlocked in the network |
| livelock | some packets are in a livelock cycle in the network |

TABLE I: **Functional bugs injected in SafeNoC.**

To analyze SafeNoC's performance impact and its ability to detect and recover from various types of design errors, we injected 11 different design bugs into our C++ implementation of SafeNoC, as described in Table I. These bugs represent possible flit-level and packet-level manifestations of functional design errors, and are based on the error model proposed in [3]. They includes misrouting of flits or entire packets, replication of flits or packets, reordering of flits within a packet, livelock and deadlock. Note that data corruption within flits is handled by the ECC already available in the baseline system.

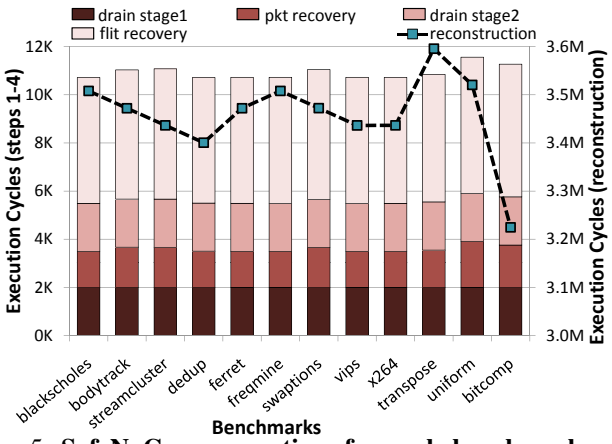


Fig. 5: **SafeNoC recovery time for each benchmark** averaged over all bugs. Execution cycles for the first 4 steps of recovery (bars-left axis) and for packet reconstruction (line-right axis).

We ran the random traffic and PARSEC workloads while triggering the bugs once per execution, and a different bug each simulation. We ran each simulation several times, each time varying when the bug is triggered, so to capture the state of the network at 10 different execution points, every 12,000 cycles after warm-up. We also repeated each experiment with 10 different random seeds for statistical confidence. With SafeNoC, all workloads complete, delivering all packets correctly to their destinations. Figure 5 reports the recovery time required by each benchmark, averaged over all random seeds, activation points and bugs, for a total of 11,000 runs. On

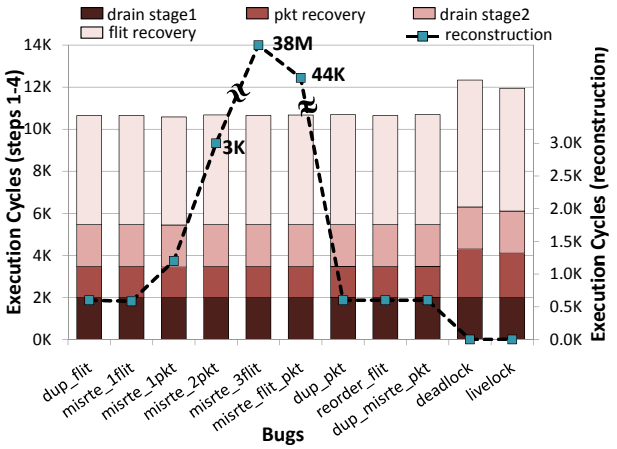


Fig. 6: **SafeNoC recovery time by bug** averaged over all benchmarks. Execution cycles for the first 4 steps of recovery (bars-left axis) and for packet reconstruction (line-right axis).

average, SafeNoC spends approximatly 11,000 cycles in the first four steps of recovery. The drain time is a preset design parameter, which, in our case, accounts for a total of 4,000 cycles. The packet recovery and flit recovery steps require on average 1,600 and 5,300 cycles, respectively. In addition, SafeNoC incurs an average of 3.4M execution cycles to reconstruct erroneous packets. The performance overhead of SafeNoC is therefore dominated by the reconstruction algorithm. In Figure 6, we analyze SafeNoC's recovery time by bug. The reconstruction time varies widely, depending on the severity of the bug and the number of flits and packets it affected. For example, bug *misrte_1flit* mixes one packet's flit among the flits of another packet. As a result, the reconstruction algorithm has two candidate flits in the system and it requires only 1,200 cycles to complete. At the opposite end, bug *misrte_2pkt* affects 32 flits in 2 different packets. Therefore, the reconstruction algorithm must consider two candidate flits for each position within the packet, requiring up to 38M execution cycles to complete. As for packet recovery, its time is constant for almost all bugs, at 1,473 cycles, required for the token to traverse all routers. *deadlock* and *livelock* are an exception: in these cases, entire packets must be retrieved from the primary network and transmitted over the checker network. For these two bugs, packet recovery requires 2,900 cycles and salvages 90 packets from the network on average. Once those packets are recovered, they no longer need reconstruction, thus the corresponding reconstruction time is 0. Finally, flit recovery time depends on the severity of the design error. The more packets are affected by the error, the more stray flits are left in the primary network, and the more must be recovered. Thus, on average, SafeNoC requires between 11K to 38M cycles to recover the system from a bug, assuming uniform clock domains on cores and NoC. For a CMP operating at 1GHz, and considering the extreme case when one design error manifests as often as every minute, the performance impact of SafeNoC is thus between $1.83 \times 10^{-5}\%$ and 0.06%.

We further investigate the relation between number of flits in error and reconstruction time by varying the number of flits misrouted in bug *misrte_1flit* from 1 to 14. During recovery, the number of flits retrieved doubles with the number of flits in error. Then, during reconstruction, there are two candidate flits for each missing flit position, leading to an exponential increase in reconstruction time. Therefore, the flit recovery time increases from 5,211 to 5,237 cycles, while reconstruction time increases from 1,200 to 4.9M execution cycles over the range considered.

| **benchmark** | **%** | **benchmark** | **%** | **benchmark** | **%** |
|---|---|---|---|---|---|
| blackscholes | 0.0 | freqmine | 0.65 | vips | 0.0 |
| bodytrack | 0.0 | swaptions | 0.77 | x264 | 0.45 |
| streamcluster | 0.52 | ferret | 0.76 | dedup | 1.54 |
| **average performance overhead: 0.52%** | | | | | |

TABLE II: **Performance overhead in the absence of bugs.**

In the absence of bugs, SafeNoC has a negligible performance overhead, as seen in Table II. This overhead is due to false positives in SafeNoC's detection, occurring when congestion in the primary network causes a data packet's delivery to be delayed and the corresponding destination counter to timeout. However, with a carefully calibrated timeout value, the occurrence of such false positives, and thus the performance impact of SafeNoC, is minimal.

| | **design** | **area** ($mm^2$) | **%** |
|---|---|---|---|
| Baseline | 8x8 mesh | 4.8 | 100 |
| SafeNoC additions | router additions | 0.15 | 3.3 |
| | NI additions | 0.18 | 3.75 |
| | checker router | 0.11 | 2.3 |
| **SafeNoC overhead over 8x8 mesh interconnect: 9.35%** | | | |
| **SafeNoC overhead over complete 64 SPARC CMP: 2.41%** | | | |

TABLE III: **SafeNoC area overhead.**

### C. Area Results

Finally, we evaluated the area overhead of SafeNoC. Our results reported in Table III indicate that SafeNoC has a 9.35% silicon area overhead over an 8x8 mesh primary network, corresponding to a 2.41% overhead over a complete CMP with 64 SPARC cores and the same baseline 8x8 NoC. We compared this overhead to a mainstream end-to-end, acknowledgement-based error recovery scheme as in [21]. Area overhead in these systems is primarily due to large data buffers needed to store the packets in-transit. We estimated the size of these buffers by monitoring the number of packets at each source waiting for acknowledgement. For an 8x8 primary network with 16-flit data packets, up to 4 data packets can be awaiting acknowledgements at a single source and correspondingly the retransmission-based system incurs an area overhead of 66.3% over the baseline network and 17.4% over the CMP. Therefore, SafeNoC offers more than a 7x improvement in area overhead compared to the commonly used retransmission-based method. Our simulations also show that SafeNoC buffer storage efficiency grows with data packet to signature compression ratio, and with a data packet size of 64 flits in a 8x8 mesh, SafeNoC uses less than a fourth of the buffer space of the retransmission-based scheme. This gain can be attributed to SafeNoC's ability to provide correctness by storing small signatures, in contrast to large data packets.

## VI. SafeNoC Design and Constraints

For an effective SafeNoC implementation, the design must be tailored to the characteristics of the baseline interconnect. The signature size is a design parameter that affects the performance of the checker network and thus needs to be appropriately selected. As explained in section III-B, we aim to have a checker network that delivers signatures to their destinations before the corresponding data packets arrive through the primary NoC, while minimizing the number of times when it lags behind. This can be achieved by choosing a suitable signature size that can identify errors with minimum aliasing, while being sufficiently small so that in most cases signatures can be transmitted on the checker network faster than data packets. In our implementation, we chose 16-bit signatures, and we validated this design decision by conducting experiments to determine how often the checker network "wins" against the primary network. We found that in this setup, at 0.38 flits/cycle per node injection rate (at the onset of saturation), for 99% of the packet-signature pairs, the signature arrives first, and in the few cases where the signature is lagging, the difference is less than 3 cycles. The packet delivery timeout and the signature buffer size are two other design parameters that need to be tuned to reduce area and performance overheads. The former determines when recovery is initiated, with large values delaying the initiation and small values resulting in false positives. To determine an appropriate packet delivery timeout, we measured the lead time of the checker network packet delivery, in a congested network with both random traffic and PARSEC benchmarks, and found 4,000 cycles to be a conservative timeout. The size of the signature buffer in the network interfaces is determined by the maximum number of outstanding signatures. On the verge of saturation, this value is 11, thus we designed our system with 12 entries to have a safety margin. In the rare event of filling all signature buffers, our system triggers a recovery.

SafeNoC can detect and recover from a wide variety of errors affecting both flits and packets, such as misrouting and reordering errors, deadlocks and livelocks, *etc.* However, it does not protect against data payload bit-level errors, since we assume that ECC is present in the primary routers. Therefore, it can not recover from design errors that result in bit-level corruptions beyond the capabilities of the ECC in-use. In addition, since SafeNoC does not retain a copy of the data in-transit, it can not recover from errors that cause flits to be dropped in transfer. One way to overcome this limitation, is to provide additional detection mechanisms to detect these errors and initiate recovery before data is dropped.

## VII. Conclusions

In this paper, we presented SafeNoC, a runtime end-to-end error detection and recovery technique to guarantee the functional correctness of CMP interconnects. SafeNoC augments the interconnect with a lightweight and simple checker network and it detects functional errors by comparing the signature of every received data packet with its look-ahead signature that was delivered through the checker network. In case of mismatches, our novel recovery approach collects blocked packets and stray flits from the primary network and distributes them over the checker network to all processor cores, where our reconstruction algorithm reassembles them. SafeNoC can detect and recover from a broad range of functional design errors, while incurring a low performance impact, requiring between 11K and 39M execution cycles to recover from an error. Our evaluation indicates that SafeNoC requires only 2.41% area overhead in a 64-core CMP system.

*Acknowledgments*

This work was developed with partial support from the Air Force Office of Scientific Research and the Gigascale Systems Research Center.

## References

[1] System Verilog Assertions. http://www.systemverilog.org/.

[2] Advanced Micro Devices, Inc. *Revision Guide for AMD Athlon(TM) 64 and AMD Opteron(TM) Processors*, Aug. 2005. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/25759.pdf.

[3] A. Alaghi, N. Karimi, M. Sedghi, and Z. Navabi. Online NoC switch fault detection and diagnosis using a high level fault model. *Proc. DFT*, 2007.

[4] K. V. Anjan and T. M. Pinkston. An efficient, fully adaptive deadlock recovery scheme: DISHA. In *Proc. ISCA*, 1995.

[5] T. M. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proc. MICRO*, 1999.

[6] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proc. ICS*, 2006.

[7] S. Bell et. al. TILE64 - processor: A 64-core SoC with mesh interconnect. In *Proc. ISSCC*, 2008.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proc. PACT*, 2008.

[9] D. Burger and T. Austin. The SimpleScalar Toolset, version 3.0.

[10] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[11] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester. Vicis: A reliable network for unreliable silicon. In *Proc. DAC*, 2009.

[12] Intel Corporation. *Intel(R) StrongARM(R) SA-1100 Microprocessor Specification Update*, Feb. 2000.

[13] Intel Corporation. *Core 2 Duo and Intel Core 2 Solo Processor for Intel Centrino Duo Processor Specification Update*, September 2007.

[14] Intel Corporation. *Intel Core i7-900 Desktop Processor Series Specification Update*, July 2010.

[15] International Business Machines Corporation. *IBM PowerPC 750GX and 750GL RISC Microprocessor Errata Notice*, July 2005.

[16] J. Kim and H. Kim. Router microarchitecture and scalability of ring topology in on-chip networks. In *Proc. NoCArc*, 2009.

[17] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston. A lightweight fault-tolerant mechanism for network-on-chip. In *Proc. NoCs*, 2008.

[18] P. Lopez, J. Martinez, and J. Duato. A very efficient distributed deadlock detection mechanism for wormhole networks. In *Proc. HPCA*, 1998.

[19] J. M. Martínez et. al. Software-based deadlock recovery technique for true fully adaptive routing in wormhole networks. In *Proc. ICPP*, 1997.

[20] A. Meixner, M. E. Bauer, and D. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. In *Proc. MICRO*, 2007.

[21] S. Murali et. al. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test*, 22(5), 2005.

[22] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunet: A cheap and robust fault-tolerant packet routing mechanism. In *Proc. ISCA*, 2004.

[23] D. Starobinski et. al. Application of network calculus to general topologies using turn-prohibition. *IEEE Trans. Networks*, 11(3), 2003.

[24] Synopsys. Synopsys Magellan. http://www.synopsys.com.

[25] I. Wagner and V. Bertacco. Engineering trust with semantic guardians. In *Proc. DATE*, 2007.

[26] Y. J. Yoon, N. Concer, M. Petracca, and L. Carloni. Virtual channels vs. multiple physical networks: A comparative analysis. In *Proc. DAC*, 2010.