Correct Runtime Operation for NoCs through Adaptive-Region Protection

Rawan Abdel-Khalek and Valeria Bertacco
Computer Science and Engineering Department, University of Michigan
(rawanak,valeria)@umich.edu

Abstract—Networks-on-chip (NoCs) are increasingly being adopted as the interconnect model for systems-on-chip and homogeneous and heterogeneous chip-multiprocessors. However, in practice, design-time verification of NoCs is always partial, due to their large scale and the challenges that hinder verification efforts. As a result, functional design bugs are bound to escape and potentially manifest at runtime, compromising system functionality. We propose REPAIR, a runtime solution to detect and recover from functional design errors that have escaped in NoCs. Existing runtime verification techniques incur significant area and performance overheads to monitor and check the correctness of every packet traversing the network. However, REPAIR relies on a retransmission-based technique that adaptively determines the subset of packets requiring protection by identifying dynamic network regions where the specific runtime execution is likely to expose design bugs. REPAIR achieves runtime correctness with significantly better overall performance: over 75% fewer acknowledgment packets relative to a traditional solution.

I. INTRODUCTION

Network-on-chip (NoC) interconnects are the central medium of communication in many current and future chip-multiprocessors (CMPs) and systems-on-chip (SoCs). Therefore, the NoC’s functional correctness is critical, and it requires extensive verification, both as a stand-alone fabric and when embedded in the complete system. Mainstream verification efforts initially target the correctness of the router architecture. However, the overall execution of the network is a result of system-level interactions that occur as responses to the traffic patterns traversing it, and thus the complete verification of the NoC requires modeling the entire network. Design-time verification still relies heavily on simulation-based approaches, whether utilized in pre-silicon verification or during verification on emulation and post-silicon platforms. However, the continuously increasing size and design complexity of NoC interconnects poses significant challenges to these verification efforts, because simulation-based verification does not scale well with large and complex systems and it is inherently incomplete. The result is that NoC-based products are released into the market with corner-case design bugs latent in the unverified executions. Those design bugs become problematic if the unverified functionality is exercised at runtime, while being utilized by the user, potentially compromising the entire system.

We propose REPAIR (Runtime Error Protection over Adaptively Identified Regions), a solution to ensure communication correctness in NoC interconnects at runtime. REPAIR targets functional design bugs that have escaped verification and that manifest at runtime in the NoC. Such bugs tend to be well-hidden and are only exposed when the runtime execution triggers a specific complex set of events or corner-cases that were never tested. Moreover, at runtime, applications running on CMPs and SoCs typically have varying traffic patterns, such that not all portions of the network encounter the same set of runtime operations, and throughout an application’s execution, some regions exhibit runtime conditions that are more likely to expose latent design bugs. REPAIR dynamically identifies those regions and protects packets traversing them. Before a packet enters an error-prone region, it is copied into the retransmission buffer of one of the routers that are at the periphery of the target region. The correct delivery of such a packet to its destination sends back an acknowledgment to the intermediate node that created the copy, which in turn frees the retransmission buffer. If the intermediate node times-out before receiving an acknowledgment, REPAIR initiates recovery, during which routers retransmit copies of the packets residing in their retransmission buffers. The manifestation of design bugs at runtime is an infrequent occurrence, and upon retransmission, the network is extremely unlikely to encounter precisely the same conditions that triggered the original bug. Figure 1 depicts a high-level overview of our solution.

Fig. 1. High-level overview of our runtime solution. REPAIR identifies network regions exhibiting operations prone to exposing latent design bugs and protects packets traversing these regions from being affected by the bugs.

II. IDENTIFYING ERROR-PRONE NETWORK REGIONS

An important aspect of our scheme is the identification of network regions that are exhibiting runtime operations exposing escaped design bugs. During development, the functional correctness of the NoC’s most common operations, including its simple and fundamental executions are typically well-validated. The network is also subjected to randomly generated traffic patterns and application traffic, to exercise its more complex functionality. However, the design space spanned by all possible NoC operations is intractable, and it is not possible to run every possible traffic configuration. In fact, it is likely that a major fraction of the bugs is hidden precisely within these complex functionalities that cannot be exhaustively validated. At runtime, various types of applications run on a NoC-based system, and they can generate previously unverified network executions. However, even during a single application’s execution, not all regions of the network encounter the same traffic patterns. Routers that have fewer in-flight packets operate within a limited subset of
the design state space that typically covers the NoC’s simpler behaviors, which are well-verified. Whereas, network regions with more activity and a larger number of contending in-flight packets exercise a wider range of complex operations. It is in those regions that we are more likely to encounter a previously unverified corner-case situation that is hiding a bug. In our solution, we rely on congestion metrics to approximate when a region is exhibiting complex activity. Congestion is not only a direct measure of the number of packets in-flight, but it also captures contention and hence the level of interactions observed within router components and across routers.

We rely on a distributive and iterative congestion detection algorithm to identify congested regions and to adapt our classification to changes in traffic, as illustrated in Figure 2.

**Step 1 - Identify locally congested routers.** Routers determine their local congestion status, based on their internal buffer occupancy, similar to the scheme described in [1]. Each router keeps track of the number of input buffer entries in-use across all of its input buffers. If the number of occupied entries exceeds a certain fraction, the router flags congestion. Once the congestion flag is set, it can only be deflagged when the buffer occupancy falls and stays below another threshold for a certain number of clock cycles.

**Step 2 - Identify congested regions.** The local congestion flag is passed to all of its direct neighbors, through a 1-bit bi-directional link added between all routers (local_cong link). Then, each router determines whether it belongs to a congested region based on two criteria: 1) Is the router itself congested? or 2) Is it neighboring two congested routers? The latter allows us to group congested routers into more contiguous regions. The result of this evaluation sets a new flag, in_cong_region, which is again transmitted to all the first-hop neighboring routers through another 1-bit bidirectional link.

**Step 3 - Identify peripheral routers.** Each router receives the in_cong_region flag from its neighbors. If at least one of its neighbors belongs to a congested region, the router determines that it is on the periphery of a congested region.

**III. ERROR DETECTION AND RECOVERY**

When REPAIR identifies a congested network region, routers at the periphery of the region and within it, initiate our acknowledgment-retransmission scheme to protect packets traversing the congested region. We refer to these routers as the initiating routers. As packets are injected into the network, they are augmented with an error-detection code. Once in-flight, packets traversing an initiating router and heading towards a congested network area are marked as ack_required and are copied into one of the router’s retransmission buffers. When an ack_required packet is delivered to its destination node, an acknowledgment is sent back to the initiating router to release the retransmission buffer. If the initiating router times-out before receiving an acknowledgment, an error is assumed to have occurred, and a recovery signal is transmitted across the network through a 1-bit serial link. The timeout functionality at initiating routers and the error-detection code that is added to every packet allow REPAIR to detect incorrect packet delivery errors, as well as data corruptions, independently of the exact location of the design bug. Upon receiving the recovery signal, each router drops all ack_required packets traversing it at the time, which clears the effects of the detected bug. Then, each router retransmits a copy of the packets residing in its retransmission buffers. Non-ack_required packets and regular network operations proceed normally in conjunction with the retransmissions. Since design bugs during runtime operation manifest rarely and are exposed only under specific execution conditions, the retransmitted packets are likely to encounter different operational conditions and timings that will not trigger the same bug to re-manifest.

**Creating packet copies at initiating routers.** REPAIR maintains a congestion status table per router that records which of the router’s output ports are congested. Moreover, we assume that the retransmission buffers at each node reside in the corresponding network interface. Therefore, once an initiating router decides to apply acknowledgment-retransmission to a particular packet, it must duplicate it and eject a copy to the local network interface. Figure 3 shows an illustration of a peripheral router applying acknowledgment-retransmission to packet P1. P1 is forwarded to its destination output port (OPx) and its copy is ejected to be stored in one of the retransmission buffers. In traversing a typical wormhole router, packets undergo four main steps: route computation (RC), virtual channel allocation (VCA), switch allocation (SA) and crossbar traversal. In the RC step, a packet’s output direction is determined. Then, in the VCA step, the router arbitrates for one of the virtual channels of the selected output port. Lastly, during SA, flits arbitrate for the crossbar and, once that is granted, proceed to the downstream router. Once a packet’s output direction is known after the RC step, a simple lookup in the congestion status table allows REPAIR to classify whether this packet requires protection. Then, in the VCA step, this packet must request a virtual channel in its assigned output port and the ejection port. The packet successfully completes VCA only if both virtual channels have been granted. If all retransmission buffers are full, a packet requiring duplication must stall. Note that REPAIR utilizes a separate virtual channel.
Fig. 3. Applying acknowledgment-retransmission at a peripheral router. Since Packet P1 is heading to the North (entering a congested area), it is protected by REPAIR. This peripheral router duplicates P1 and forwards a copy to be stored in one of the retransmission buffers.

to eject packet copies to the retransmission buffers. Similarly, during the SA step, the flits of the packet requiring protection should simultaneously request two crossbar connections, one to its desired output port and another to the ejection port. When both connections are granted, the original flits proceed to their destination output port, and a copy of each flit is ejected to be stored in a retransmission buffer.

IV. EXPERIMENTAL EVALUATION

We modeled an 8x8 mesh interconnect using the cycle accurate Booksim simulator [2] and implemented REPAIR. We also generated directed random traffic workloads, summarized in Figure 4, to model communication patterns of applications running on CMPs and SoCs, where depending on the type of applications and their scheduling, some nodes communicate more frequently than others. In each generated workload, the first and third phases model periods of low-activity, where all nodes are injecting packets at a uniform and low rate. The second phase models a more active communication period, where a certain number of randomly chosen node-pairs communicate more frequently than others. In the workloads labeled mc, we set 6 high-communication node-pairs, chosen at random, to send traffic at a high injection rate, while other nodes inject uniform traffic at a much lower rate. The generated mc workloads model traffic patterns with medium-sized congested regions spanning 7 up to 20 routers, on average. In the workloads labeled hc, we assume 10 randomly chosen high-communication node-pairs, creating larger congested regions, consisting of 16 up to 35 routers, on average. We utilize different random seeds to generate 10 mc and 5 hc workloads, each with a different set of high-communication pairs.

A. Bug Detection and Recovery

We modeled five bugs, each triggered when routers encounter a different set of complex conditions due to several contending packets traversing it. As soon as the conditions are met, the corresponding router drops one of its packets. This experimental framework prevents the correct delivery of a packet upon a bug manifestation, which models any type of error that could occur due to a bug, and not just dropped-packet errors. Table I describes our bugs and their manifestation conditions. We then ran our 15 workloads and observed the total number of times each bug manifested across all workloads and the number of workloads that were affected by the bug.

B. Network Performance

We also evaluated the network performance when using REPAIR and compared it to a traditional source-based retransmission technique. In source-based retransmission, a copy of every packet is stored in a retransmission buffer at the source node, before the packet is injected into the network. As such, source-based retransmission is typically adopted to ensure correctness in the presence of communication errors such as packet corruptions and dropped packets. Figure 5 shows the execution time of our workloads when using REPAIR and source-based retransmission, normalized to the execution time of a baseline system without any protection capabilities. In both source-based and REPAIR, we assume two retransmission buffers per node. A network equipped with REPAIR can achieve communication correctness with 60% faster execution times than source-based retransmission. Source-based retransmission causes, on average, a 75% slowdown, due to

![Diagram of a router with retransmission buffers and congestion status table]
the reduction in throughput, as many packets are stalled at injection waiting for free retransmission buffers. Whereas, in REPAIR, there is less contention for the retransmission buffers at each node, since only a small fraction of packets require protection (on average, 25% of total packets injected).

C. Implementation overhead

To identify congested regions, each router is equipped with a counter to track buffer occupancy and registers to store the threshold values and the congestion status table. With a 5 input-port router, 2 virtual channels per port, and 8-flit buffers, 7 bits are needed for the occupancy counter and each of the threshold values. The congestion status table consists of 1 entry to store the router’s local congestion status and 4 entries to indicate the congestion status of the neighboring routers. The congestion status itself can be encoded with 1 bit. The total storage overhead required is 116 bits, less than that of one flit-entry of a router’s input buffer. The control logic to update the occupancy counter and compare it to the threshold values is negligible (<0.1%). As for creating copies of in-flight packets requiring protection, we modeled this functionality and synthesized the modified and baseline router using Synopsys Design Compiler with a 45nm target library and found that the area overhead is 0.86%. In total, the area overhead of implementing REPAIR is <1%.

V. RELATED WORK

Ensuring the runtime correctness of communication in NoCs is a widely explored topic. However, many solutions focus on correctness in the face of transient or permanent faults in the NoC hardware. [3] is a recent survey of common fault-tolerance techniques. Source-based acknowledgment-retransmission mechanisms are among those that are most relevant to our work [4]. However, we show that relying on source-based retransmission for detecting and recovering from escaped design bugs is prohibitively costly and ineffective to resolve design bugs. We propose a different acknowledgment-retransmission approach that successfully protects only the packets that are susceptible to design bugs.

Several works have proposed runtime approaches that target functional design errors in processors [5]–[7], whereas a few focus on NoCs, [8], [9]. The authors of [8] use a combination of formal verification and runtime checkers to ensure correct NoC operations. [9] is solely based on a runtime solution, but fails to protect the network from some of types of errors. Both approaches rely on augmenting the baseline network with a checker network, introducing significant area overheads (>9%). In contrast, REPAIR relies on an extremely lightweight technique that can successfully protect network communication, independently of the types of errors encountered, while incurring (<1%) area overhead.

Many previous works, [1], [10]–[13], have explored congestion detection in NoCs and they rely on a variety of metrics including crossbar demand, free virtual channels, free buffer space, output port contention, etc. Local congestion is then propagated through the network, by aggregation, [10], or through a separate subnetwork [11], [12]. In these solutions, congestion detection is used for providing better performance, routing algorithms, or congestion control mechanisms [13], [14], often requiring an accurate global view of congestion. However, in REPAIR, we aim to detect congested regions for the purpose of identifying areas with complex activity, thus we resort to a simpler, yet effective, estimate of congestion.

VI. CONCLUSIONS

We introduced REPAIR, a runtime solution that protects on-chip networks from the manifestation of design bugs. Complex execution scenarios are often encountered in high-traffic network regions, making these regions susceptible to the manifestation of latent design bugs. REPAIR identifies congested regions and protects packets traversing them with an acknowledgment-retransmission scheme. Unlike common source-based retransmission techniques, acknowledgment-retransmission is selectively utilized only for a small subset of packets, while allowing REPAIR to successfully detect and recover from errors that manifest in congested regions.

REFERENCES