Providing Formal Latency Guarantees for ARQ-based Protocols in Networks-on-Chip

Eberle A. Rambo, Selma Saidi, and Rolf Ernst
Institute of Computer and Network Engineering
TU Braunschweig, Germany
{rambo|saidi|ernst}@ida.ing.tu-bs.de

***Abstract*—Networks-on-Chip (NoCs) are the backbone of Multiprocessor Systems-on-Chip (MPSoCs). In this paper, we perform a formal worst-case communication time analysis of Automatic Repeat reQuest (ARQ) protocols for NoCs. Therefor, we integrate the transport layer analysis for general networks and the network layer analysis for NoCs. An ARQ variant optimized for DMA transfers (DMA ARQ) is introduced and analyzed. Experimental evaluation with Stop-and-Wait, Go-Back-N, and DMA ARQ, in the context of real-time memory traffic is presented, including both error-free and error cases. DMA ARQ achieves a factor 6 improvement on latency bounds over conventional Stop-and-Wait.**

I. Introduction

Networks-on-Chip are the replacement to the non-scalable bus interconnect in Multiprocessor Systems-on-Chip (MPSoCs). In NoCs, memory operations and I/O are packets traversing a network. As any network and as the central component in the MPSoC, the NoC must handle noise and errors that may occur in its infrastructure in order to provide reliable communication [1]. These errors can be of transient nature, called soft errors, or permanent nature, the hard errors. We focus on the former.

Soft errors are caused by process variability, alpha particle strikes, cosmic radiation and crosstalking. In the NoC, they present a variety of effects depending on where they occur [1]. This work focuses on soft errors affecting data, resulting in packet corruption, loss, or delayed delivery. It is therefor assumed that errors affecting the control of the network are appropriately handled in the lower network layers. Error detection and retransmission techniques, such as Cyclic Redundancy Check (CRC) and Automatic Repeat reQuest (ARQ), are used to ensure data integrity and packet delivery, and are implemented in the data-link (hop-by-hop) and/or transport layers (end-to-end) [2]. We focus on ARQ protocols operating in the transport layer.

ARQ-based protocols are widely used to guarantee packet delivery [2]. In its simplest variant Stop-and-Wait, for each packet sent, the sender waits for an acknowledgement (ACK) from the receiver before sending the next packet or retransmitting after a timeout, in case of error. The (visibly limited) throughput is improved in Go-Back-N [2]. It allows n packets to be sent (the *send window*) before stopping and waiting for an acknowledgement.

An illustrative example is show in Figure 1. Go-Back-N sends $n = 3$ packets at time ① before waiting for an ACK. The receiver acknowledges the successful delivery of each packet ②. In ③, packet 7 is not successfully delivered. After a timeout t_{out}, the sender retransmits all unacknowledged packets ④. Out-of-order and duplicated packets are discarded. Notice that Stop-and-Wait is a case of Go-Back-N with $n = 1$.

This work has been partially funded by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 – spp1500.itec.kit.edu).

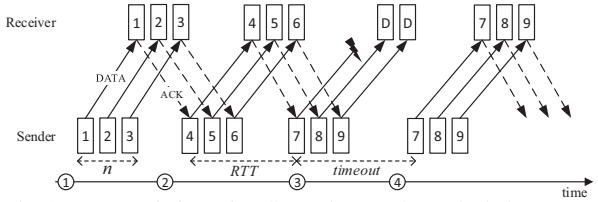


Fig. 1. A transmission using Go-Back-N ARQ, send window $n = 3$.

Currently, NoC-based MPSoC platforms are being evaluated for real-time and safety-critical embedded systems [3]. Such systems must provide guarantees that tasks *respond in time* and *reliably*. As basis for any response time guarantee, formal real-time analysis methods are used to derive upper and lower bounds for throughput and latency of the NoC [4]. Transport protocols must also be timely analyzed to evaluate the impact (i.e delay) incurred on the communication and to guarantee that the specified timing requirements are still met both on the error-free and error cases. To the best of our knowledge, no formal analysis of ARQ-based transport protocols for Networks-on-Chip has been proposed so far.

In this paper, we perform a formal analysis of ARQ-based protocols for NoCs. Therefore, we provide the integration of transport and network layer analyses for Networks-on-Chip and evaluate the performance of typical memory traffic in real-time systems. Stop-and-Wait and Go-Back-N ARQ perform well with general memory traffic without bursts. The performance of Direct Memory Access (DMA) transfers however are drastically impaired by these protocols. To overcome this gap efficiently, we propose a variation of Go-Back-N optimized for DMA, called DMA ARQ. The protocol exploits the characteristics of DMA transfers to achieve high throughput and avoid large and expensive retransmission buffers. In our experiments, the protocol was able to improve formal latency bounds of DMAs by a factor of 6 over Stop-and-Wait.

The **contribution** of this paper is three-fold: **(1)** the integration of ARQ analysis for general networks [5] with NoC analysis; **(2)** a protocol optimized for DMA transfers in NoCs with formal analysis for both error-free and error cases; and **(3)** the evaluation of ARQ protocols' performance in a real-time context for error-free and error cases. Without loss of generality, the paper considers [4] as the underlying NoC analysis. Since the approach is based on Compositional Performance Analysis (CPA) [6], it can be easily composed with other packet- and wormhole-switched network models.

II. Related Work

Fault tolerance for NoCs has been constantly investigated throughout the years [7]. Many mechanisms have been proposed and compared in terms of power and area overheads, and their resulting reliability improvement. [8], [9] investigate the energy-reliability trade-off between different soft error

detection and recovery strategies at the data-link layer. [10] and [11] present analyses of NoC designs with respect to performance, reliability and energy perspective. They allow good comparison between different design choices and error recovery schemes, in both data-link and transport layers. The latency estimations however are either based on queueing theory [10] or simulation [11], and thus cannot provide the bounds required in the real-time domain [12].

In real-time systems, formal performance analyses play an important role by guaranteeing responsiveness in time. For networks, formalisms such as Network Calculus and Compositional Performance Analysis (CPA) are employed to provide worst-case communication time bounds [6], [12]. A formal communication time analysis of wormhole-switched NoCs with two-stage arbitration was proposed in [13]. Based on CPA, the analysis provides latency bounds for individual traffic streams on the network. The work has been extended in [4] to support the sharing of virtual channels by traffic streams. The analysis is valid in the real-time domain but has the implicit assumption that no errors affect the network, and hence, the transmission.

ARQ-based protocols are widely used in all kinds of networks to guarantee the delivery of data [2]. They can be applied both in the data-link and transport layers. ARQ-based protocols Stop-and-Wait and Go-Back-N have been formally analyzed in [5] for general packet-switched networks. The CPA based analysis models the protocols on a transport layer and considers both the error-free and the error cases. The analysis however cannot be directly applied to NoCs, whose more advanced designs feature wormhole-switching. In those, packets are composed of flits and the arbitration is performed at flit granularity [4]. The integration of the transport layer analysis, operating on a packet basis, with the NoC analysis, operating on a flit basis, is required. To the best of our knowledge, no formal analysis has addressed the analysis of ARQ-based protocols for Networks-on-Chip.

III. Integrating Transport and Network Analyses

Transport protocols, such as DMA ARQ and Go-Back-N ARQ, introduce additional flow control to the communication. This comes from packets that are retained e.g. until acknowledgements from previous packets are received (handshaking) or until a timeout occurs (retransmission). It creates a circular dependency: the performance of the transport layer depends on the network latency, which in turn depends on the traffic injected by the transport layer. We solve the problem of providing formal performance bounds for such networks using CPA, which allows easy integration of network and transport layer analyses.

A. Compositional Performance Analysis

CPA relies on independent local analyses of the system resources, such as router ports and CPUs, and a global analysis loop that aggregates the local results [6]. It provides worst-case response times and jitter of tasks. The system model is based on *resources* providing services, *tasks* consuming these services, and *event models* specifying task activation patterns. Task activations are triggered by an external source or by events propagated from other tasks (predecessor tasks). The activations in an event model are given by event arrival curves $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, which return the minimum and maximum number of events that can arrive in a given time interval Δt. Their pseudo-inverse counterparts $\delta^+(q)$ and $\delta^-(q)$ return the maximum and minimum time interval between the first and last events in any sequence of q event arrivals. A conversion method is presented in [14].

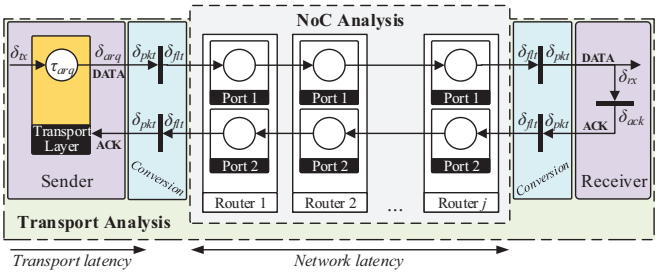


Fig. 2. Modeling of transport layer protocol and the underlying NoC in CPA.

The analysis is carried out in a local step and global loop. In the local step, the local analysis uses the busy window approach [15] to derive each task's response time and output event model. In a global loop, the analysis propagates tasks' output event models to their dependent tasks (activated by the completion of other tasks), becoming then their input event models. The analysis stops when all event models are stable (a fix point is reached) or when predefined constraints are violated (e.g. maximum response time).

B. Transport and Network Modeling

The analysis problem is modeled in CPA based on [5] and shown in Figure 2. The analysis' starting point is the traffic stream producing packets according to a given packet-based event model δ_{tx}. The packets are handled at the transport layer by an arbitrary transport protocol. The protocol is modeled as a single task τ_{arq}, whose response time reflects the protocol behavior. The transport layer then injects traffic in the NoC router according to the output event model δ_{arq} of that task.

As illustrated in Figure 2, a protocol with handshaking is a bidirectional communication stream. In the NoC, it is mapped as two unidirectional streams: one for the data (sender $\rightarrow$ receiver) and the other acknowledgements (receiver $\rightarrow$ sender). The NoC is modeled in CPA based on [4]: each output port of a router is mapped as a resource, and traffic streams are chains of tasks mapped to resources. Resource arbitration depends on the router arbitration. Details available in [4].

The transport layer operates on a packet granularity, and so does its analysis (δ_{tx} and δ_{arq} model packet arrivals). The analysis of the NoC however can be performed on a packet or flit granularity, depending on whether the network is packet- or wormhole-switched. Thus, an appropriate conversion between the two domains is required (shown in Figure 2 between sender and NoC, and between NoC and receiver). Equation 1 converts from a packet-based event model $\delta^-_{pkt,i}(q)$ to a flit-based one $\delta^-_{flt,i}(q)$. Index i refers to a traffic stream i.

$$\delta^-_{flt,i}(q) = \max\Big\{(q-1)\cdot d_{min}\,, \\ \delta^-_{pkt,i}\big(\lceil q \div size_i \rceil\big) + [(q-1) \bmod size_i]\cdot d_{min}\Big\} \quad (1)$$

where $size_i$ is the size of any data packet in stream i (in flits), and d_{min} is the minimum distance between two consecutive flits [4]. The q-th flit activation depends on the activation of the packet containing the q-th flit plus an offset depending on the flit's position in the packet (second term). The first term enforces the minimum distance between q flit activations. For $q \leq 1$, $\delta^-_{flt,i}(q) = 0$.

Equation 2 provides the flit-to-packet conversion.

$$\delta^-_{pkt,i}(q) = \delta^-_{flt,i}\big((q-1)\cdot size_i + 1\big) \quad (2)$$

The q-th packet activation depends on the activation of the first flit of the q-th packet. For $q \leq 1$, $\delta^-_{pkt,i}(q) = 0$.

The acknowledgements (ACKs) are injected back in the network by the receiver δ_{ack} (righthand side in Figure 2). Protocols acknowledging each packet, such as Stop-and-Wait, inject ACKs in the network when a packet arrives (i.e. $\delta_{ack} = \delta_{rx}$). Protocols that acknowledge blocks of packets must perform a model conversion from δ_{rx} to δ_{ack}.

$$\delta^-_{ack,i}(q) = \delta^-_{rx,i}\big((q-1) \cdot g_i + 1\big) \tag{3}$$

where g_i is the number of packets that the protocol receives before sending an ACK. For $q \leq 1$, $\delta^-_{ack,i}(q) = 0$.

The analysis is then carried out in *two* local steps and a global loop: extending the regular CPA flow (cf. Section III-A) with one additional step for the transport layer. First, the NoC analysis computes the worst-case network latency (1). Then, the transport analysis derives the worst-case transport latency and updates the output event models accordingly (2). The global loop proceeds as usual in CPA.

The transport layer analysis and derived metrics (e.g. latency), naturally, depend on the modeled protocol. The analysis and metrics of DMA ARQ are introduced in Section IV; Stop-and-Wait and Go-Back-N ARQ in [5].

The underlying NoC analysis is arbitrary. Therefore, we define the interface between the transport and network analyses.

C. Interface with the Underlying Network-on-Chip Analysis

The inputs for the NoC are the event models $\delta_{arq,i}$ and $\delta_{ack,i}$, which model the traffic in the data stream i and its ACK stream i_{ack}, respectively (after appropriate conversion).

The output of the underlying NoC analysis is used as an input to the transport layer analysis in the form of best- and worst-case **round trip time** (RTT_i^- and RTT_i^+). They are lower and upper bounds on the time it takes for any packet in a stream i to be sent through the network and acknowledged. First, let us define the worst-case latency L_i^+ for any packet in a stream i:

$$L_i^+ = l_i^+(size_i) \tag{4}$$

where $l_i^+(size_i)$ is the worst-case latency to transmit $size_i$ flits in the NoC [4].

RTT_i^+ is then obtained by the sum of the worst-case latency of a data packet L_i^+ and the worst-case latency of its acknowledgement $L_{i_{ack}}^+$, as follows:

$$RTT_i^+ = L_i^+ + L_{i_{ack}}^+ \tag{5}$$

RTT_i^- is calculated similarly, but using best-case latencies.

IV. The DMA ARQ Protocol

A. Protocol Description

Go-Back-N ARQ performs well for traffic without bursts. With bursts, as in the case of DMA transfers, it requires larger send windows to deliver good performance [2], implying large retransmission buffers and, thus, area and power overheads [11]. To achieve lower latencies with reduced area overhead, we propose an ARQ protocol optimized for DMAs, the DMA ARQ. The protocol is a variation of Go-Back-N, where the send window is increased to match the transfer size. In case of errors, packets are selectively repeated, similar to Selective Repeat [2]. Its implementation is enabled by exploiting two properties found in DMA transfers.

First, we exploit the fact that the number of packets in a DMA transfer, the transfer size n_{dma}, is known beforehand. This allows the protocol to acknowledge the whole transfer instead of each packet. As a result, throughput is not limited by the protocol. In case the transfer was not successful, a negative-acknowledgement (NACK) is sent, selectively requesting the retransmission of missing and corrupt packets. Integer packets can be processed and have their contents forwarded to the memory controller as soon as they arrive, since the memory address of each packet's contents can be inferred from the transfer's initial address.

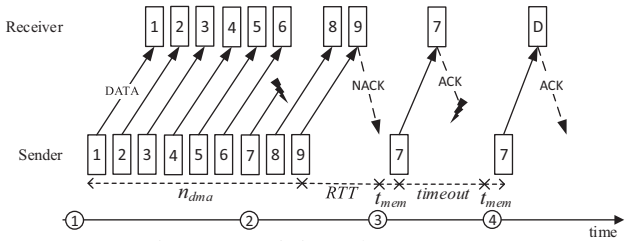


Fig. 3. Transmission under DMA ARQ.

Second, due to the nature of DMA transfers, it is not necessary to keep packets in a retransmission buffer. In case of retransmission, the data from missing or corrupt packets can be read from the sender's local memory. This is only possible for DMA traffic. In general traffic, the data is not accessible. An example is a non-blocking uncached write issued by a processor. The protocol implementation avoids overhead in area but increases the overhead in time in case of retransmission, requiring an additional time t_{mem} to read from the local memory. Furthermore, the protocol assumes that memory consistency is guaranteed independently by a mechanism in software or hardware, a standard procedure.

An illustrative example is show in Figure 3. The sender starts a transfer with size $n_{dma} = 9$ ①. All packets are immediately forwarded. Eventually, a packet is not successfully delivered ②, and the receiver sends a NACK at the end of the transfer. Its content is read from the local memory and packet 7 is retransmitted ③. The packet is received but the acknowledgement is lost. After a timeout ④, the sender retransmits the last packet of the transfer. The duplicate packet is dropped. The receiver acknowledges and the transfer ends.

The worst-case communication time analysis of DMA ARQ is presented in two parts. First, we model the protocol behavior in the error-free scenario. The analysis models the data transfer itself, corresponding to a write DMA transfer (it differs from a read in the request starting the transfer). Then, we evaluate the worst-case impact of errors on a transmission and incorporate it into the analysis. We assume that the network interface itself is not faulty and that it is able to detect packet corruption.

B. Formal Analysis: the error-free case

To calculate the latency, we need to derive the worst-case DMA ARQ delay $R^+_{dma,i}$, the largest period of time in which a packet is retained by the protocol. Therefore, we derive first the **busy period** $w_{dma,i}$ for the protocol [15]. The largest time interval $w_{dma,i}$ in which packets arrive at the transport layer and need to be queued while the DMA ARQ protocol is waiting for ACKs of previous packets is given by

$$w_{dma,i} = \left\lfloor \frac{\eta^+_{tx,i}(w_{dma,i}) - 1}{n_{dma}} \right\rfloor \cdot RTT_i^+ \tag{6}$$

In DMA ARQ, all packets are transmitted as soon as they are generated. The only moment requiring an acknowledgement is in the end of each transaction, and it takes at most RTT_i^+. Equation 6 forms an integer fixed point problem ($w_{dma,i}$ on

both sides), typical in busy-window based analyses. It can be solved iteratively, starting with a very small ϵ ($w_{dma,i} = \epsilon$).

Now we bound the largest time interval to forward a sequence of q packets under error-free conditions, the **worst-case multiple packet forwarding time** $B^+_{dma,i}(q)$. Considering that the first packet is starting the transfer, the $B^+_{dma,i}(q)$ of a stream i is given by:

$$B^+_{dma,i}(q) = \left\lfloor \frac{q-1}{n_{dma}} \right\rfloor \cdot RTT^+_i \tag{7}$$

All packets in a transmission can be transmitted as soon as they are available. Except for the first packet of a subsequent transmission. At the end of a transmission, the protocol waits for an ACK before allowing the next transfer to start. This takes time RTT^+_i for each complete transmission.

The **best-case multiple packet forwarding time** $B^-_{dma,i}(q)$ is the smallest time interval to forward q packets. Considering that the first packet is starting the transfer, it is given by:

$$B^-_{dma,i}(q) = \left\lfloor \frac{q-1}{n_{dma}} \right\rfloor \cdot RTT^-_i \tag{8}$$

The reasoning is the same as for Equation 7, but considering the best-case round-trip latency RTT^-_i.

The **worst-case response time** $R^+_{dma,i}$ is the largest time interval that any packet is delayed by DMA ARQ before being forwarded to the network. $R^+_{dma,i}$ of a stream i is bounded by:

$$R^+_{dma,i} = \max_{1 \le q \le \eta^+_{tx,i}(w_{dma,i})} \{B^+_{dma,i}(q) - \delta^-_{tx,i}(q)\} \tag{9}$$

The response time of the q-th packet is the time between its arrival ($\delta^-_{tx,i}(q)$) and its injection in the network ($B^+_{dma,i}(q)$). As in fixed-priority scheduling, the maximum can be found among packets arriving during the busy period $w_{dma,i}$ [5].

The event model capturing the traffic injection in the network by DMA ARQ can now be derived. The **output event model** $\delta^-_{dma,tx}$ propagated by the transport layer without errors is bounded by:

$$\delta^-_{dma,tx}(q) = \max\{\delta^-_{tx}(q) - R^+_{dma} + R^-_{dma}, B^-_{dma}(q-1)\} \tag{10}$$

The minimum distance between q packets injected in the network is the original packet flow minus the jitter introduced by the protocol ($R^-_{dma} - R^+_{dma}$). The maximum function ensures that a packet cannot be forwarded before the previous one ($B^-_{dma}(q-1)$).

Let us now bound the time it takes to complete a transfer using DMA ARQ. The **overall latency** $\mathbb{L}^+_{dma,i}(q)$ of a transfer comprising q data packets in a stream i is given by:

$$\mathbb{L}^+_{dma,i}(q) = \delta^-_{tx,i}(q) + L^+_i + R^+_{dma,i} \tag{11}$$

In the worst-case, the latency consists of the time it takes to inject the q packets, plus the latency it takes for the last (q-th) packet to be delivered by the network, plus the worst-case delay for that packet introduced by the protocol ($R^+_{dma,i}$). Due to causality (packets cannot pass each other), all previous packets must have been received by the time the last packet is received. In case of one complete transfer, $q = n_{dma}$.

C. Formal Analysis: the error case

The analysis of the previous section assumed an error-free scenario. Now we remove this assumption and analyze scenarios where a number of errors affect the transmission, the k-error scenario.

Errors can have several types of effects, such as latency increase, packet corruption, and packet loss. Packet corruption causes the packet to be eventually discarded by the receiver, after applying e.g. CRC. The latency increase is assumed to be smaller than the time it takes to retransmit a lost packet. Therefore, the analysis considers packet loss as the worst-case impact of an error.

The worst-case impact of a packet loss in a transfer under the DMA ARQ protocol is when the acknowledgement is lost right before being delivered, similar to Go-Back-N [5]. It causes the maximum load in the network and the maximum delay: a retransmission of the last packet (local memory access t_{mem} plus RTT^+_i) is triggered after a timeout t_{out}. Similarly, a second error causes a second retransmission after timeout. The resulting additional time is $k \cdot (t_{out} + t_{mem} + RTT^+_i)$, for k errors. We can reason that this scenario results in the worst-case latency. Let us assume that the errors cause regular data packet losses and a retransmission in the end of the transfer. This results in an upper bound of $k \cdot (t_{mem} + RTT^+_i)$. Since $k \cdot (t_{out} + t_{mem} + RTT^+_i) > k \cdot (t_{mem} + RTT^+_i)$ and $t_{out} > 0$, our worst-case error latency holds. This assumes that the loss of any data packet has the same impact and does not cause the whole transmission to fail.

First, we integrate the impact of errors into the busy period. The **k-error busy period** $w_{dma,i}(k)$ is is given by:

$$w_{dma,i}(k) = k \cdot (t_{err} + RTT^+_i) + \left\lfloor \frac{\eta^+_{tx,i}(w_{dma,i}(k)) - 1}{n_{dma}} \right\rfloor \cdot RTT^+_i \tag{12}$$

where $t_{err} = t_{out} + t_{mem}$. In the worst case, in addition to the error-free busy window, each error leads to a timeout t_{out} and a retransmission RTT^+_i (data packet and ACK).

Similarly, we derive the **k-error worst-case multiple packet forwarding time** $B^+_{dma,i}(q,k)$, accounting for the impact of k errors on the transfer.

$$B^+_{dma,i}(q,k) = k \cdot (t_{err} + RTT^+_i) + \left\lfloor \frac{q-1}{n_{dma}} \right\rfloor \cdot RTT^+_i \tag{13}$$

In addition to $B^+_{dma,i}$, in the worst case, each error leads to a timeout and a retransmission whose impact is also bounded by $t_{out} + RTT^+_i$, as in Equation 12.

The **k-error worst-case response time** $R^+_{dma,i}(q)$ is derived from Eq. 9 to account for the k-error versions of $w_{dma,i}(k)$ and $B^+_{dma,i}(q,k)$:

$$R^+_{dma,i}(k) = \max_{1 \le q \le \eta^+_{tx,i}(w_{dma,i}(k))} \{B^+_{dma,i}(q,k) - \delta^-_{tx,i}(q)\} \tag{14}$$

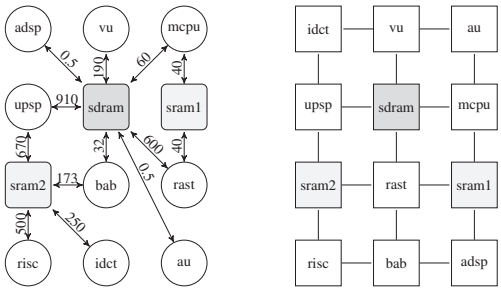
Finally, we bound the time it takes to complete a transfer using DMA ARQ under errors. The **k-error overall latency** $\mathbb{L}^+_{dma,i}(q,k)$ of a transfer comprising q data packets in a stream i is given by:

$$\mathbb{L}^+_{dma,i}(q,k) = \delta^-_{tx,i}(q) + L^+_i + R^+_{dma,i}(k) \tag{15}$$

Besides the latency in the error-free case (Equation 11), in the k-error scenario, Equation 15 must account for the latency overhead. This is included by the last term, the k-error worst-case response time $R^+_{dma,i}(k)$.

V. Experiments

In our experiments we evaluate, in a real-time context, the performance of memory traffic of the MPEG decoder



(a) MPEG decoder. (b) Mapping on 3x4 NoC.

Fig. 4. The node graph annotated with communication bandwidth requirements (MB/s), and mapping of the MPEG decoder application.

(Figure 4) on a NoC implementing end-to-end Stop-and-Wait, Go-Back-N, and DMA ARQ. Firstly, we analyze the performance of the general memory traffic and compare it with simulation results. Then, we evaluate the performance of the memory traffic when parts of it consist of DMA transfers. The analyses were implemented in PyCPA [14]. The simulations were carried out on the OMNeT++ network simulator [16].

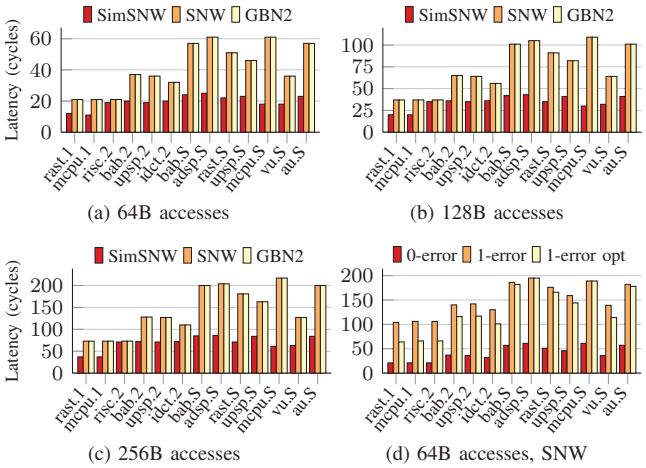
The employed NoC consists of 3x4 nodes in a 2D-mesh topology. Each node consists of a tile, a router, and a network interface connecting both. Links connect the routers and network interfaces. The NoC implements wormhole-switching, virtual channel flow control, XY source routing and SLIP arbitration in the routers. The frequency of operation is 800MHz. We utilize [4] as the underlying NoC analysis, with overhead parameters per hop O_r and per path O_p set to 0. The analysis of [5] was used for Stop-and-Wait and Go-Back-N.

The MPEG decoder [17] employed in the experiments is presented in Figure 4a, which shows the node graph and the bandwidth requirements (MB/s) between the communicating nodes. In the experiments, the data flows from the memories to the nodes. The nodes have been mapped to a 3x4 NoC as depicted in Figure 4b [18]. Each arrow represents the source and destination of a *traffic stream*. We refer to a stream by "node.X", where "node" is the processing element and "X" is the memory: 1 for *sram1*, 2 for *sram2*, and S for *sdram*. For instance, *rast* has two streams: *rast.1* and *rast.S*.

A. General traffic

In the first experiment, we evaluate the performance of traffic streams consisting of periodic, non-bursty accesses to memory. We assume that the traffic in real-time systems exhibits well understood and predictable patterns e.g. streaming applications [17] or predictable execution models such as superblocks [19], [20]. Therefore, we vary the size of the data accesses of each MPEG node in three scenarios: 64, 128 and 256Bytes transported in a packet with 5, 9 and 17 flits, respectively. The frequency of the data accesses is derived to match the specified bandwidth. For instance, 500 MB/s (*risc.2*) corresponds to 64B every 128ns (102 cycles at 800MHz).

Figures 5a, b and c show the maximum observed and analytical worst-case latencies for a packet in a given traffic stream for increasing access sizes. For Stop-and-Wait, analysis (SNW) provides valid bounds to simulation (SimSNW) for all accesses sizes. The margin over simulation is noticeably higher for traffic to/from the *sdram*. This comes from the underlying NoC analysis: all traffic streams coexist in the NoC in this setup, and share the same network interface to access the *sdram*, figuring a bottleneck. Moreover, the values from simulation are observed maximums. The true worst-case



(a) 64B accesses (b) 128B accesses

(c) 256B accesses (d) 64B accesses, SNW

Fig. 5. Maximum end-to-end packet latencies for Stop-and-Wait (observed in simulation SimSNW, and analytical SNW) and Go-Back-N with $n = 2$ (analytical GBN2). Error-free scenarios in (a,b,c), error scenario in (d).

lies somewhere between the values from the analysis and the observed ones.

Switching from Stop-and-Wait to Go-Back-N $n = 2$ (GBN2) and further increasing the send window ($n > 2$, not shown in the figures) does not decrease latency. This type of traffic is characterized by the *quasi* absence of bursts. As a result, small retransmission buffers can be employed for general traffic without compromising performance. Not only that, but if the distance between two packets being injected in the network is smaller that the round trip time, Go-Back-N has no gain over Stop-and-Wait, and a single-slot retransmission buffer can be employed. However, this decision can critically affect the performance of bursty traffic, such as DMA transfers.

The impact of errors on transmissions is shown in Figure 5d, for 64B accesses and Stop-and-Wait. The Figure shows the worst-case latency when (1-error) employing a safe, equal timeout t_{out} for all streams, and when (1-error opt) optimizing the timeout individually per stream, slightly larger than its latency bound. Since the worst-case retransmission is largely dominated by the timeout, optimizing it results in up to 38% lower latencies in a 1-error scenario.

B. DMA traffic

In the second experiment, we analyze the performance of bursty memory traffic, characteristic of DMA transfers. This type of traffic is predominant in safety-critical MPSoCs platforms [19]. Therefore, we assume that nodes with high throughput to the *sdram* use DMA to be more efficient (*rast.S*, *upsp.S*, and *vu.S*). We evaluate scenarios with 4KB, 8KB and 16KB transfers. Their frequency are set to match the specified bandwidth. Each transfer consists of 32, 64, and 128 packets, respectively for each scenario. The packets are 9-flit long. We assume that the DMA transfers are synchronized in software or hardware, such that one transfer is performed at a time. This reflects real hardware limitation and programming model. The non-DMA streams are configured as 64B, Stop-and-Wait general traffic.

Figures 6a, b and c show the maximum latency bounds for entire DMA transfers in the error-free case. For transfers of same size e.g. 8KB, maximum latency varies more than 300%. The extra delay is caused by background traffic, specially by the stream from *upsp.S*, and is captured by the underlying NoC

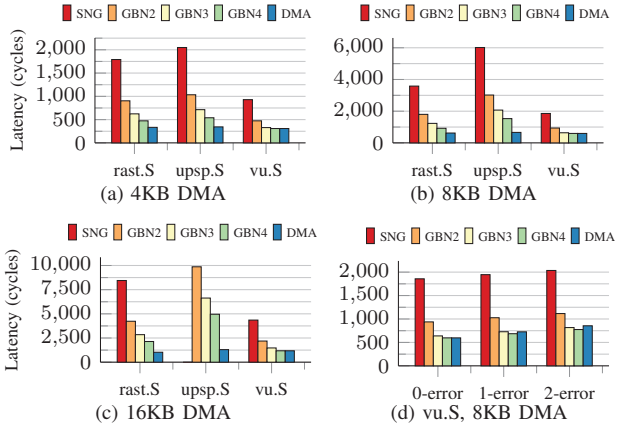


Fig. 6. Worst-case end-to-end latency of DMA transfers for Stop-and-Wait (SNW) and Go-Back-N with $n = 2$, 3 and 4 (GBN2, GBN3, GBN4) and DMA ARQ (DMA). Error-free scenarios in (a,b,c), error scenarios in (d).

analysis. Stop-and-Wait is not suitable for bursty transfers, resulting in very high latencies and in the unschedulability of *upsp.S* with 16KB transfers (Figure 6c). Go-Back-N improves the performance as its send window is increased. Nonetheless, DMA ARQ presents the lowest latency bound in all cases. Only for *vu.S*, due to very low network interference, Go-Back-N is able to achieve the same performance at the expense of a large retransmission buffer (send window 4). Despite the latency variations for the DMAs, the latency bounds for the non-DMA streams (not shown in the Figure) remain the same when varying the protocols.

When considering error scenarios, DMA ARQ outperforms all protocol configurations except for stream *vu.S*, detailed in Figure 6d. The analysis considers $t_{out} = 60$ and $t_{mem} = 40$ cycles. This is explained by the fact that DMA ARQ requires extra time to read from the local memory when retransmitting. Since, in the error-free case (0-error), GBN3, GBN4 and DMA deliver similar latency bounds, DMA presents slightly higher latencies in error scenarios (5% higher than GBN4 in 1-error).

Now let us evaluate the impact of the different transport protocols. Figure 7 shows the latency improvement, area overhead, and error overhead when varying the protocol. The values are normalized to a 3x4 NoC with Stop-and-Wait. Area overhead metric covers retransmission buffer requirements. Extending the send window in Go-Back-N introduces more area overhead than it increases performance, specially for smaller transfer sizes. Notice that the area overhead will scale with the system, i.e. the values in the Figure are multiplied by the number of nodes in the NoC. Notice also that, for DMA and general traffic to transmit in parallel, a node would require at least two protocol instances. Error overhead measures the additional time required to finish the transfer in a 1-error scenario. In the worst-case, it takes Go-Back-N the same time as Stop-and-Wait. DMA ARQ, on the other hand, presents a major performance improvement (6.6x on average) while avoiding additional buffers. This comes at the price of additional time overhead when recovering from an error, since the data has to be read from the local memory instead of a retransmission buffer. This penalty however is only seen during error occurrences and does not increase with transfer length.

VI. Conclusion

In this paper, we performed a formal communication time analysis of Automatic Repeat reQuest (ARQ) protocols for Networks-on-Chip. Therefore, we presented an integration

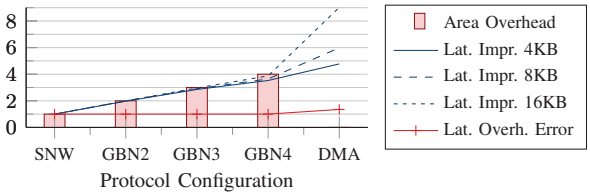


Fig. 7. Comparison of protocols' performance, relative to Stop-and-Wait.

between the formal analysis of the transport layer and the analysis of the network layer by means of event model conversion between the packet and flit domains. We have also presented DMA ARQ, a variation of Go-Back-N ARQ optimized for DMA transfers, with formal analysis. Our experimental evaluation includes Stop-and-Wait, Go-Back-N, and DMA ARQ for typical memory traffic scenarios found in real-time systems. Results show that Stop-and-Wait suffices for general memory traffic but not for DMA traffic, where DMA ARQ provides latency bounds 6.6x lower. This analysis can be used to provide formal latency guarantees for on-chip traffic in real-time systems. The analysis can be easily composed with different protocols and NoC models.

References

[1] E. A. Rambo, A. Tschiene, J. Diemer, L. Ahrendts, and R. Ernst, "FMEA-Based Analysis of a Network-on-Chip for Mixed-Critical Systems," in *Proc. of NOCS'14*, 2014.
[2] A. Tanenbaum and D. Wetherall, *Computer Networks*. Pearson Prentice Hall, 2011.
[3] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic, "IDAMC: A Many-Core Platform with Run-Time Monitoring for Mixed-Criticality," in *Proc. of HASE 2012*, 2012.
[4] E. A. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in *Proc. of DATE'15*, 2015.
[5] P. Axer, D. Thiele, and R. Ernst, "Formal timing analysis of automatic repeat request for switched real-time networks," in *Proc. of SIES'14*, 2014.
[6] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis–the SymTA/S Approach," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, 2005.
[7] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in networks on chip," *ACM Computing Surveys*, vol. 44, 2012.
[8] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *Transactions on CAD, IEEE*, vol. 24, no. 6, 2005.
[9] A. Ejlali, B. Al-Hashimi, P. Rosinger, S. Miremadi, and L. Benini, "Performability/energy tradeoff in error-control schemes for on-chip networks," *Transactions on VLSI, IEEE*, vol. 18, no. 1, jan. 2010.
[10] J. Kim, D. Park, C. Nicopoulos, N. Vijaykrishnan, and C. Das, "Design and analysis of an NoC architecture from performance, reliability and energy perspective," in *Proc. of ANCS 2005*, 2005.
[11] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks on chips," *Design Test of Computers, IEEE*, vol. 22, no. 5, 2005.
[12] A. E. Kiasari, A. Jantsch, and Z. Lu, "Mathematical formalisms for performance evaluation of Networks-on-Chip," *ACM Computing Surveys*, vol. 45, no. 3, 2013.
[13] J. Diemer, J. Rox, M. Negrean, S. Stein, and R. Ernst, "Real-Time Communication Analysis for Networks with Two-Stage Arbitration," in *Proc. of EMSOFT'11*, 2011.
[14] J. Diemer, P. Axer, and R. Ernst, "Compositional performance analysis in python with PyCPA," *Proc. of WATERS*, 2012.
[15] K. Tindell, A. Burns, and A. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Systems*, vol. 6, no. 2, 1994.
[16] A. Varga *et al.*, "The OMNeT++ discrete event simulation system," in *Proc. of the European Simulation Multiconference, ESM'2001*, 2001.
[17] E. B. Van Der Tol and E. G. Jaspers, "Mapping of MPEG-4 decoding on a flexible architecture platform," in *Electronic Imaging 2002*. International Society for Optics and Photonics, 2001.
[18] S. Murali, D. Atienza, L. Benini, and G. De Micheli, "A method for routing packets across multiple paths in NoCs with in-order delivery and fault-tolerance gaurantees," *VLSi DeSign*, 2007.
[19] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing analysis for TDMA arbitration in resource sharing systems," in *Proc. of RTAS'10*, 2010.
[20] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for COTS-based embedded systems," in *Proc. of RTAS'11*, 2011.