Using Segmentation to Improve Schedulability of Real-Time Traffic over RRA-based NoCs *

Meng Liu, Matthias Becker, Moris Behnam, Thomas Nolte
Mälardalen University, Västerås, Sweden
Email: {meng.liu, matthias.becker, moris.behnam, thomas.nolte}@mdh.se

ABSTRACT

Network-on-Chip (NoC) is the interconnect of choice for many-core processors and system-on-chips in general. Most of the existing NoC designs focus on the performance with respect to average throughput, which makes them less applicable for real-time applications especially when applications have hard timing requirements on the worst-case scenarios. In this paper, we focus on a Round-Robin Arbitration (RRA) based wormhole-switched NoC which is a common architecture used in most of the existing implementations. We propose a novel segmentation algorithm targeting RRA-based NoCs in order to improve the schedulability of real-time traffic without modifying the hardware architecture. According to the evaluation results, the proposed segmentation solution can significantly improve the schedulability of the whole network.

1. INTRODUCTION

The complexity of today's embedded applications is steadily increasing. Many industrial domains face the shift away from single-core platforms towards multi-and many-core platforms in order to fulfill the applications growing demand for computational power, while still satisfying the requirements on low power consumption.

The interconnect of choice on such platforms is the Network-on-Chip (NoC) [2]. Typically a many-core platform is arranged into a number of nodes, where each node can have one or multiple cores as well as local memory. A node contains further a network interface which connects to a NoC router. These routers are in turn connected to each other and they thus comprise the NoC. Wormhole-switching is implemented in most of the existing NoC designs [8]. In contrast to store-and-forward switching, wormhole-switching requires significantly smaller buffers on each router. This is the case since a packet is transmitted in so called flow control digits (flits), the elementary unit of transmission on a NoC. A header flit is transmitted first through the network. As long as the next link on the path is free and the buffer on the next router can accommodate at least one flit, the header continues its transmission. The remaining flits follow in a pipelined manner. During this transmission, the flits of one packet can span over multiple routers, hence the name wormhole-switching.

Our current work targets applications with mixed timing requirements. A subset of tasks and packets has strict timing requirements (called *real-time traffic*), while others have no timing requirements at all (called *best-effort traffic*). One industrial example for such a system is a Programmable Logic Controller (PLC) which is used to control processes in factory automation. Apart from the actual control algorithms which are time critical, many such devices allow a user to connect to the system via web-services in order to parameterize or observe the process variables. These parts do not face any timing requirements. However, since the NoC is shared among real-time and best-effort traffic, the timing analysis, which is required to obtain the Worst-Case Traversal Time (WCTT) of real-time packets, must consider all packets [5]. This means that a system designer must specify parameters such as packet size and period for best-effort traffic as well. Due to the system characteristics, prior knowledge of such information about best-effort traffic might be difficult or even not possible. Thus, a method to integrate both real-time and best-effort traffic on the NoC while satisfying all the timing requirements is of importance for many applications.

In order to provide real-time guarantees, many network/flow control mechanisms have been proposed (e.g. [6][9][10][4]). However, most of these solutions require support from specific hardware designs. Consequently, for many of the existing implementations, those solutions cannot be directly applied. In this paper, we propose a novel segmentation algorithm for RRA-based NoCs, which aims to improve the schedulability of all the real-time traffic in a NoC. The proposed algorithm can be used at software level such that no modification of hardware is required.

The contributions of this work are:

- A novel packet segmentation algorithm is proposed. The algorithm aims to improve the schedulability of real-time traffic without modifying the hardware design.
- Extensive evaluations have been performed. According to the evaluation results, applying the proposed segmentation approach can significantly improve the schedulability of the whole network.

1.1 Related Work

The predictability of timing behavior is important for real-time applications. Targeting such type of applications, a number of research works have been presented in the literature, such as time-triggered NoCs (e.g. [6][9][10]), the Back Suction flow-control scheme [4], flow regulation [3], and fixed-priority based NoCs [11][12]. These solutions resolve the scheduling problem of real-time traffic in NoCs from different aspects, however, they either require specific hardware support or runtime monitoring of traffic. For example, a time-division multiplexing based solution typically requires routers that can support time-slot based transmission, and synchronization among all the nodes in the network has to be carefully considered. A fixed-priority based NoC requires implementation of multiple virtual-channels in order to achieve preemptions between different priority levels, which can also result in high buffer cost. In this paper, we address the timeliness issue in NoCs from another perspective - packet segmentation. The proposed segmentation algorithm is an off-line solution, and it can be simply applied on most of the existing

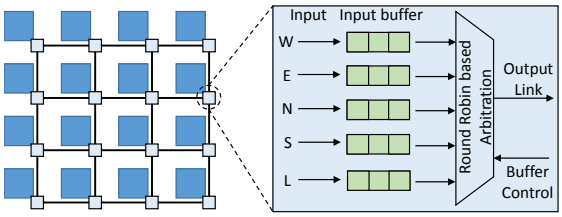

Figure 1: The abstracted architecture of an example NoC.

Commercial-Off-The-Shelf (COTS) implementations without requiring any hardware modification.

The remainder of the paper is organized as follows. In Section 2 the system model is introduced. In Section 3, we illustrate the segmentation approach for real-time traffic in RRA-based NoCs. The evaluations are presented in Section 4, and Section 5 concludes the paper.

2. SYSTEM MODEL

In this paper, we focus on an $m \times m$ 2D-mesh based wormhole-switched NoC using XY-routing (as shown in Figure 1). Such type of design has been utilized in many existing NoC implementations (e.g. [1][13]). At each router, a Round-Robin Arbitration (RRA) mechanism is used to control the link access. Under such a mechanism, each input buffer can deliver at most one packet to the output link within one round-robin cycle.

The network contains a set of real-time flows (denoted as S^{rt}). A flow is a series of packets with the same characteristics. Each real-time flow f_i is generated periodically or sporadically, and it can be characterized by $f_i = \{L_i, T_i, D_i, Sr_i, Ds_i\}$. L_i represents the size of a complete packet of f_i. T_i is the period of a periodic flow or the minimum inter-arrival time of a sporadic flow. Each real-time flow has a relative deadline D_i. A flow is defined as schedulable if its WCTT (denoted as W_i) is no larger than its deadline (i.e. $W_i \leq D_i$). The network is defined as schedulable if all the real-time flows meet their deadlines. Moreover, each flow has a fixed path/route which starts from its source node Sr_i and ends at its destination node Ds_i.

3. SEGMENTATION FOR NOCS WITH REAL-TIME TRAFFIC

In this paper, we focus on RRA-based NoCs containing only real-time traffic. A motivation example is shown in Figure 2, where the NoC contains 5 real-time flows. Now we consider f_1 as the flow under analysis. We notice that, f_1 can get direct blocking from f_2, f_3 and f_5 because of shared links, and indirect blocking from f_4 because of back-pressure on f_2. Therefore, the transmission of all the other 4 flows can affect the WCTT of f_1 (i.e. W_1). Assume that f_1 misses its deadline (i.e. $W_1 > D_1$). In order to make f_1 meet its deadline, W_1, which consists of the basic transmission time (i.e. the transmission time without any blocking) and blocking delay from other flows, has to be reduced. For most applications, the basic transmission time of a flow is difficult to decrease, since it requires reduced data payload. Therefore, reducing the blocking caused by other flows is more practical. In this section, we propose a packet segmentation based solution to reduce the blocking that a flow may experience during its transmission, such that its worst-case traversal delay can be decreased accordingly.

Assume that f_i is the flow under analysis, and that k occurrences of f_j are involved in W_i. An RRA-based NoC typically uses a non-preemptive policy. The length of each occurrence of blocking from f_j then depends on the size of a complete packet of f_j. In other words, the packet size of f_j can directly affect W_i. Under the RRA policy, without changing the period of f_j, the number of occurrences of f_j considered in W_i is fixed. Therefore, if we can decrease the length of each occurrence of blocking from f_j, W_i can be reduced accordingly. To achieve this, we propose to apply packet segmentation on f_j. Packet segmentation means dividing a packet into a number of sub-packets. After the segmentation, f_j may cause less blocking to f_i because the length of each blocking occurrence is reduced. However, since each sub-packet has to wait for one round-robin cycle at a router, f_j itself may thus experience more blocking which can result in a larger W_j. Thus, a smaller sub-packet of f_j can provide shorter W_i by inducing less blocking, which on the other hand can result in a larger W_j by containing a higher number of sub-packets. Therefore, the selection of sub-packet sizes needs to be considered carefully.

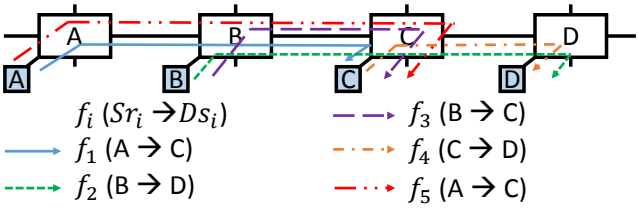

Figure 2: An example of five real-time flows in a NoC.

Alg. 1 Segmentation for real-time flows

```
1:  Input: S^rt
2:  UnschedulableFlows ← ∅
3:  for all f_i in S^rt do
4:      W_i ← ComputeWorstCaseLatency(f_i)
5:      if W_i > D_i then
6:          add f_i into UnschedulableFlows
7:      end if
8:  end for
9:  for all f_j in UnschedulableFlows do
10:     result ← Alg. 2 (f_j)
11:     if result = UNSCHEDULABLE then
12:         return UNSCHEDULABLE
13:     end if
14: end for
15: return SCHEDULABLE
```

The complete segmentation approach is presented in Alg. 1. First, the algorithm checks the schedulability of the flow set without applying the segmentation process (Alg. 1, line 3-8). Note that, to compute a packets WCTT, the applied schedulability test must consider that a packet may be transmitted in multiple segments. If all the flows can meet their deadlines, the algorithm can directly terminate since the timing requirement is already fulfilled while each packet keeps its original payload size. On the other hand, if the timing analysis shows that certain flows cannot meet their deadlines, the algorithm starts to apply a segmentation process in order to save these flows from missing their deadlines (Alg. 1, line 9-14).

The segmentation process is presented in Alg. 2, where f_i is the flow which may miss its deadline without segmentations. In order to reduce the blocking caused to f_i, we only need to consider the flows in S_i^I (representing all the flows which have potential to affect W_i). Furthermore, not all the flows in S_i^I actually contribute to W_i. For example, in Figure 2, both f_2 and f_3 are included in S_1^I since both of them can cause blocking to f_1. However, f_2 and f_3 come

Alg. 2 Segmentation to make f_i meet its deadline

```
1: Input: f_i
2: while S_i^IA ≠ ∅ do
3:   for all f_j in S_i^IA do
4:     k ← L_j / L_j^S
5:     while True do
6:       pre_k ← k
7:       Increase(k)
8:       L_j^S ← L_j / k
9:       if L_j^S < σ then
10:        L_j^S ← L_j / pre_k
11:        break
12:      end if
13:      valid ← True
14:      for all f_m in Θ_j ∪ {f_j} do
15:        analyze f_m using the current L_j^S
16:        if W_m > D_m then
17:          valid ← False
18:          break
19:        end if
20:      end for
21:      if valid = False then
22:        L_j^S ← L_j / pre_k
23:        break
24:      end if
25:      analyze f_i using the current L_j^S
26:      if W_i ≤ D_i then
27:        return SCHEDULABLE
28:      end if
29:    end while
30:  end for
31:  update S_i^IA
32: end while
33: return UNSCHEDULABLE
```

from the same source node, therefore, they cannot cause blocking to f_1 at the same time due to the RRA policy. An analysis thus only takes the flow which causes the maximum blocking into account. We use S_i^{IA}, which is a subset of S_i^I, to denote the set of flows which can actually contribute to the current W_i. Accordingly, in order to reduce the blocking involved in W_i, we only need to apply the segmentation process on the flows in S_i^{IA} (Alg. 2, line 3 - 30).

While segmenting a flow f_j ($f_j \in S_i^{IA}$), we increase the number of segments gradually, and the size of one segment decreases accordingly (Alg. 2, line 7 - 8). The increasing function can be a step function to keep it simple or a binary search to make it more efficient. For each given segment size, we need to check if it is acceptable (Alg. 2, line 9 to 24). When the size of one segment[1] (denoted by L_j^S) becomes smaller than the size of a single flit (denoted by σ), such a segment size is not acceptable since a flit is already the minimum transmission unit in wormhole-switched NoCs (Alg. 2, line 9 - 11). As discussed earlier, increasing the number of segments of f_j may increase the blocking caused to f_j. If the scheduling policy used at the output-port of the source node is also RRA, we only need to recheck the schedulability of f_j itself since other flows cannot get increased blocking by the reduced L_j^S. However, such a situation may not be practical for most cases. When a packet is generated by a source task, it is inserted into the buffer at the source node which typically uses a FIFO mechanism. The RRA policy, where the access of the output-link switches between different packets within one local buffer, is difficult to implement and causes extra overhead. In a more general case, the output-port at the source node simply uses a FIFO mechanism. Consequently, the real-time flows which are generated from the same source node of f_j (denoted by Θ_j) may also be affected by the segmentation of f_j. This is because the extra blocking caused to f_j can also delay the transmission of these flows if they are pipelined behind f_j at the source node. Therefore, given a new segment size of f_j, we need to recheck the schedulability of both f_j and all the flows from the same source node of f_j (Alg. 2, line 14 - 20). If the new segment size makes any of the above flows miss its deadline, such a segment size is not acceptable.

If the given segment size is not acceptable, the algorithm stops the segmentation process on f_j and starts to investigate another flow in S_i^{IA} (Alg. 2, line 11, 23). On the other hand, if the given segment size is acceptable, we can use it to reanalyze f_i. If f_i becomes schedulable, the algorithm terminates with a success (Alg. 2, line 25 to 27). Otherwise, the algorithm continues by increasing the number of segmentations of f_j or other flows in S_i^{IA}.

Note that the flows included in S_i^{IA} may vary during the segmentation process. For example, before the segmentation, f_j causes the maximum blocking to f_i at a certain router. After the segmentation, the blocking caused by f_j decreases. Consequently, the maximum blocking at the same router can be caused by another flow f_m ($f_m \in S_i^I$). In this case, in the following segmentation process, f_m which is not considered in the initial S_i^{IA} should also be taken into account. Therefore, the flow set S_i^{IA} needs to be updated continuously (Alg. 2, line 31). While updating S_i^{IA}, the flows which cannot be segmented any more will be removed. When there is no flow remaining in S_i^{IA}, the algorithm terminates since no more flows can be segmented. In this case, the algorithm returns a result of failure, which means that segmentation cannot save f_i from missing its deadline (Alg. 2, line 33).

As presented earlier, while segmenting a certain flow, the schedulability of all the flows which may be affected by the segmentation is examined. Only if all the affected flows are schedulable, the segmentation becomes effective. Therefore, we can guarantee that if a flow is schedulable in a NoC without segmentation, it remains schedulable after applying the segmentation policy; on the other hand, if a flow misses its deadline in a framework without segmentation, it is potentially schedulable using the segmentation mechanism. In other words, the proposed segmentation algorithm achieves either better or equal but never worse performance compared to NoCs without segmentation.

The segmentation process can be accomplished at the software level on each computing core. One solution is to add a middleware (an intermediate software program) between software partitions and network interfaces on each core. The middleware divides each packet into a number of sub-packets based on selected segment sizes. Each segment has its own header flit which contains the same information as included in the header of the original packet. Therefore, on the hardware level, segmented packets are simply treated as normal NoC packets (i.e. no hardware modification is required).

4. EVALUATION

In this section, we present the evaluation results of the proposed

[1]Note that a new header is added to each segment so that a segment can be transmitted as normal packets. A header is typically one single flit, which is much smaller than the data payload. To simplify the presentation, the size of the header flit is included in L_j^S.

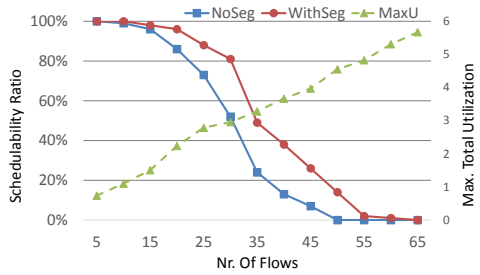

Figure 3: NoSeg vs. WithSeg. Packet size from [5, 25] flits. Flow utilization from [0.003, 0.1].

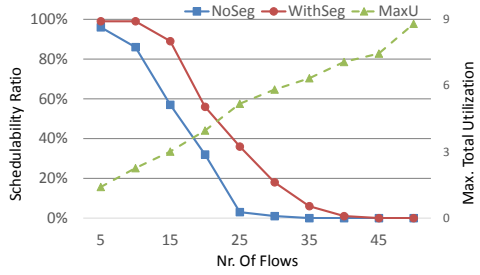

Figure 4: NoSeg vs. WithSeg. Packet size from [5, 25] flits. Flow utilization from [0.01, 0.2].

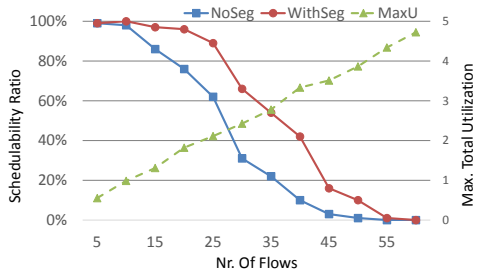

Figure 5: NoSeg vs. WithSeg. Packet size from [5, 50] flits. Flow utilization from [0.003, 0.1].

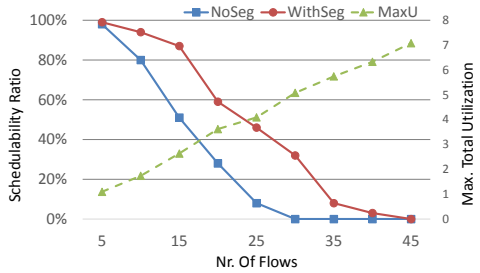

Figure 6: NoSeg vs. WithSeg. Packet size from [5, 50] flits. Flow utilization from [0.01, 0.2].

segmentation algorithm for real-time packets (i.e. Alg. 1). The evaluation uses an 8×8 2D-meshed NoC with the XY-routing. The source and destination of each flow are randomly selected but not set to the same node, otherwise communication through NoC is unnecessary. The overhead incurred by the segmentation policy (i.e. extra headers) have been taken into account. For each experiment, we create two frameworks with the same setting (including network architecture and flow set), and the only difference between these frameworks is that one framework uses packet segmentation while the other one does not. Then we examine the schedulability of both frameworks. To achieve a fair comparison, the schedulability test TRC [7] is utilized for both frameworks. The evaluation results are represented by the schedulability ratio[2] achieved by the two frameworks with respect to the number of flows in the network.

Six groups of experiments have been generated. In the first group, the packet size of each flow is randomly[3] generated from a range of $[5, 25]$ flits. The utilization of each flow is randomly selected from $[0.003, 0.1]$. The number of flows in the network are selected from 5 to 65 with a step of 5. For each setting, we generate 100 experiments. The results are shown in Figure 3. When the network only contains 5 flows, the schedulability ratio achieved by the framework with the segmentation policy (marked by *WithSeg* in the figure) is 100%, while the framework without segmentation (marked by *NoSeg* in the figure) achieves the same schedulability ratio. As the number of flows goes up, the schedulability ratios of both frameworks decrease since the total utilization of the flow set[4] increases (e.g., as shown in Figure 3, as the number of flows goes up from 5 to 65, the maximum observed total utilization increases from 0.73 to 5.67). However, the schedulability ratio of the framework without segmentation decreases obviously faster compared to the other framework. When the number of flows is 30, the schedulability ratio of the framework with segmentation is 81%, which is 29% higher than the ratio achieved by the framework without segmentation. When the number of flows reaches 50 (where the maximum observed total utilization is 4.55), the schedulability ratio of the framework with segmentation decreases to 14%, while the ratio achieved by the other framework drops to 0.

In the second group of experiments, we evaluate the frameworks with higher utilization. The generation of packet sizes remain the same as in the first group. Now the utilization of each flow is randomly selected from $[0.01, 0.2]$. The number of flows increases from 5 to 50 with a granularity of 5. The results are presented in Figure 4. Similar to the observation from the first group of experiments, as the number of flows increases, the schedulability ratio achieved by the framework with segmentation declines obviously slower than the framework without segmentation. When the number of flows goes up from 5 to 30, the schedulability ratio of the framework with segmentation decreases from 99% to 18%, while the schedulability ratio of the other framework drops from 96% to 1%. When the number of flows is 35 (where the maximum observed total utilization is 6.33), the schedulability ratio of the framework without segmentation becomes 0, while the framework with segmentation still has a schedulability ratio of 6%. By comparing the first two groups of experiments, we can observe that changing utilization of each real-time flow does not really affect the benefit (in the sense of improving schedulability) yielded by the segmentation approach.

In the third group of experiments, we investigate NoCs with larger packet sizes. Now the packet size of each flow is randomly generated from $[5, 50]$ flits, and the utilization of each flow is randomly selected from $[0.003, 0.1]$. The number of flows increases from 5 to 60 with a step of 5. As shown in Figure 5, similar to the previous results, as the total network utilization increases, the schedulability ratio achieved by the framework without segmentation decreases obviously faster than the framework with segmentation. When the number of flows is 30, the schedulability ratio of the framework with segmentation is 66%, which is 35% higher than

[2]Given an experiment setting, the schedulability ratio is the percentage of schedulable flow sets among all the generated sets.

[3]In our experiments, all the randomly generated values are selected from the given range following a uniform distribution.

[4]The total utilization of a flow set is the summation of the utilizations of all the included flows.

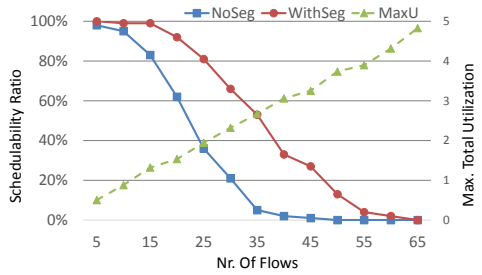

Figure 7: NoSeg vs. WithSeg. Packet size from [5, 100] flits. Flow utilization from [0.003, 0.1].

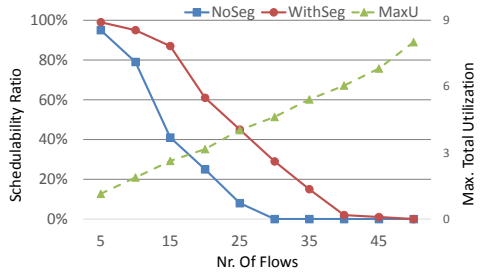

Figure 8: NoSeg vs. WithSeg. Packet size from [5, 100] flits. Flow utilization from [0.01, 0.2].

the ratio achieved by the framework without segmentation. When the number of flows reaches 50 (where the maximum observed total utilization is 3.86), only 1% of the generated flow sets are schedulable for the framework without segmentation, while the other framework still has a schedulability ratio of 10%. The schedulability ratio of the framework with segmentation becomes 0 when the number of flows reaches 60 (where the maximum observed total utilization is 4.72). The fourth group of experiments utilize the same setting as used in the third group to generate packet sizes of real-time flows, while the utilization of each flow is randomly selected from $[0.01, 0.2]$. As shown in Figure 6, the maximum difference between the schedulability ratios achieved by these two frameworks is 38% (when the network contains 25 flows). When the number of flows reaches 30 (where the maximum observed total utilization is 5.08), the schedulability ratio achieved by the framework without segmentation becomes 0. However, for the other framework, schedulable flow sets can be observed until the number of flows reaches 45 (where the maximum observed total utilization is 7.07).

Additionally, we also generate two groups of experiments with further increased packet sizes. In these two groups of experiments, the packet size of each flow is randomly selected from $[5, 100]$ flits. The flow utilization is generated in the same manner as used in the previous experiments. The results are given in Figure 7 and Figure 8, from which we can obtain very similar observations as acquired in the other experiments regarding the improvement accomplished by the segmentation approach. We notice that the maximum difference between the schedulability ratios achieved by these two frameworks is 48% (when the NoC contains 35 flows) in the fifth group (see Figure 7) and 46% (when the NoC contains 15 flows) in the sixth group (see Figure 8).

By comparing all six groups of experiments, we can observe that the improvement achieved by the segmentation approach increases as packet size goes up. For example, for the settings with flow utilization from [0.003, 0.1] (i.e. Figure 3, Figure 5 and Figure 7), as the maximum packet size increases from 25 to 100, the average improvement raises from 11% to 20% and the maximum observed improvement increases from 29% to 48%.

Generally, according to the above evaluation outcomes, we can conclude that using the proposed segmentation approach can significantly improve the schedulability of NoCs with real-time traffic. We can also observe that the improvement can always be clearly observed regardless different packet sizes and flow utilizations. Furthermore, the segmentation solution achieves more improvement when flows have larger packets.

5. CONCLUSION AND FUTURE WORKS

In this paper, we introduce a segmentation-based approach in order to improve the schedulability of real-time traffic in RRA-based NoCs. According to the evaluation results, the proposed segmentation solution can significantly improve the schedulability of the whole network.

In our ongoing work, we plan to solve the problem of transmitting both real-time traffic and best-effort traffic in the same NoC. The aim is to provide low latency for best-effort traffic while the schedulability of all the real-time traffic is still guaranteed. Moreover, in this work, we assume that all the flows are already mapped in the NoC. We can observe that the mapping of flows definitely affect the WCTTs of flows which further affects the schedulability of the whole network. Therefore, we would also like to address the mapping problem in the context of NoCs with segmentation.

6. REFERENCES

[1] Adapteva Inc. *Epiphany Architecture Reference*, 2012.

[2] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 2002.

[3] B. D. De Dinechin, Y. Durand, D. Van Amstel, and A. Ghiti. Guaranteed services of the noc of a manycore processor. In *International Workshop on Network on Chip Architectures*. ACM, 2014.

[4] J. Diemer and R. Ernst. Back suction: Service guarantees for latency-sensitive on-chip networks. In *NOCS*. IEEE, 2010.

[5] T. Ferrandiz, F. Frances, and C. Fraboul. A method of computation for worst-case delay analysis on spacewire networks. In *SIES*, 2009.

[6] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: concepts, architectures, and implementations. *Design & Test of Computers*, 2005.

[7] M. Liu, M. Becker, M. Behnam, and T. Nolte. Improving schedulability of real-time traffic over wormhole-switched nocs by applying segmentations. Technical report, MDH, 2016.

[8] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 1993.

[9] C. Paukovits and H. Kopetz. Concepts of switching in the time-triggered network-on-chip. In *RTCSA*. IEEE, 2008.

[10] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *NOCS*. IEEE, 2012.

[11] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *NOCS*, 2008.

[12] Z. Shi and A. Burns. Real-time communication analysis with a priority share policy in on-chip networks. In *ECRTS*. IEEE, 2009.

[13] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 2007.