Parallel Probe Based Dynamic Connection Setup in TDM NoCs

Shaoteng Liu (liu2@kth.se), Axel Jantsch (axel@kth.se) and Zhonghai Lu (zhonghai@kth.se)
KTH Royal Institute of Technology

***Abstract*—We propose a Time-Division Multiplexing (TDM) based connection oriented NoC with a novel double time-wheel router architecture combined with a run-time parallel probing setup method. In comparison with traditional TDM connection setup methods, our design has the following advantages: (1) it allocates paths and time slots at run-time; (2) it is fast with predictable and bounded setup latency; (3) it avoids additional resources (no auxiliary network or central processor to find and manage connections); (4) it is fully distributed and therefore it scales nicely with network size.**

Compared to a packet based setup method, our probe based design can reduce path setup delay by 81% and increase network load by 110% in an 8x8 mesh, while avoiding the auxiliary network. Compared to a centralized method, our solution can double the success rate, while eliminating the central resource for path setup and reducing the wire overhead. Synthesis results suggest that our design is faster and smaller than all comparable solutions.

I. Introduction

Circuit switching (CS) is frequently adopted for guaranteed data transfer, since it is a cost-effective technique in real-time communication [1]. CS means that resources are allocated exclusively to a particular connection for its entire lifetime. Since the exclusive allocation of a link is very inflexible and potentially blocks other communications, in the on-chip context two main variants have been explored: (1) in *Time-Division-Multiplexing (TDM)* based CS, resources are allocated exclusively only in specific time slots, while the other time slots of a finite, repeating time window can be used by other communications [2]; (2) in *Spatial-Division-Multiplexing (SDM)*, only part of a link with its corresponding buffers are exclusively allocated to a connection, while the remaining wires of the link can be used by other communications [3]. Because SDM link sharing introduces large crossbars with low clock frequency and high hardware costs, TDM based CS is more popular and e.g. used in Æthereal [1], dAElite [2], Nostrum [4], and so forth.

A main challenge for TDM based CS NoCs is to set up a contention free path and to allocate the time slots. There are two categories of techniques for path setup: one is static scheduling [5]–[7], the other is dynamic (run-time) searching. Static methods schedule connections at compile time, based on the premise that all communication requirements are known beforehand. Thus, they are not well fit for applications like H.264 with requirements for dynamic communication setups or dynamic mixes of applications.

Thus, we concentrate on dynamic setup in TDM NoCs and propose a probe based dynamic path searching method with guaranteed setup delay. In particular, our contributions are:

- We propose a probe based setup method for TDM based CS NoC. Our method does not resort to an auxiliary network for connection configuration and imposes no limitation on search algorithms.
- We present a double orientation time-wheel technique to enable two-way communication in the probe based setup. A slot-table is shared by both forward and backward messages.
- We implement a parallel probing search to find a free path. This algorithm guarantees that, if a shortest path is available, it is found within $2 \times D + K + 6$ time slots, where D is the distance between source and destination, and K is the total number of time slots in a slot table.

Taking all together, our design provides shorter critical timing path, less wire overhead, shorter setup delay and higher success rate at lower hardware cost than any previously known method.

II. Background and related work

Dynamic path searching methods can be divided into centralized [2], [8] and distributed solutions [9], [10]. In centralized solutions a special node is used to schedule all the connections in the network. This coordinating node can be a processor or a special hardware accelerator. Path scheduling algorithms running on hardware accelerators such as HAGAR [8] are about 100-1000 times faster than running as software on a processor [11], [12]. However, centralized setup methods suffer from the lack of scalability. As the network grows, the coordinator node becomes the bottleneck [8], [12]. Also, since retrying of failed requests causes the blockage of the following requests, failed setup requests are usually dropped in centralized setup methods. Furthermore, centralized solutions often depend on an additional network for delivering the configuration information to the routers.

Traditionally, distributed solutions in TDM NoCs are implemented with configuration packets [9]. Configuration packets such as setup, tear-down and Ack/Nack, require a separate *Best Effort (BE)* network during the connection establishment procedure. This approach suffers from three major drawbacks. Firstly, using an additional BE network for the connection setup is an unnecessary overhead. Secondly, the routing algorithms have to be deterministic to ensure setup, tear-down and Ack/Nack packets of a connection are on the same route so that the booked slots inside the routers along the route can be read/removed correctly, thus significantly restricting the path searching space. Thirdly, compared with our probing search, tear-down and Ack/Nack signals have to be sent in the form of packets. These packets are often underutilized and contend with setup and other packets. There is no delay guarantee for configuration packets, rendering the setup delay unpredictable.

Another kind of distributed path searching method is probe based searching, although it was only used in CS NoCs without TDM or SDM link sharing. For NoCs, the concept of probing was firstly proposed by Wiklund et al. [13]. Pham et al. [10] developed a backtracking path searching algorithm, which has better performance than Wiklund's. Liu et al [14] developed a parallel probing method for CS NoC. It can complete a search over all possible shortest paths within $O(D)$ time complexity where D is the geometric distance between source and destination. They demonstrated superior performance of this parallel

978-3-9815370-2-4/DATE14/ © 2014 EDAA

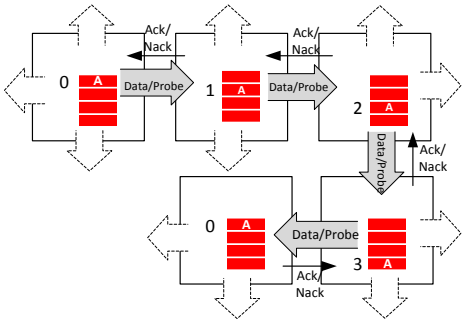


Fig. 1. The usage of double orientation time wheel

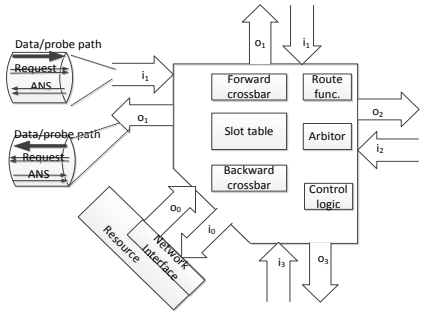


Fig. 2. Overview of the router

probing algorithm compared to Pham's backtracking algorithm. We adapt Liu's parallel probing algorithm for TDM based NoCs.

Another track of work uses per-connection *Virtual Channels (VCs)* and round-robin arbitration to share links and provide communication guarantees (e.g. [15], [16]). VCs are expensive resources, since they consist of buffers, multiplexers, demultiplexers and require separate flow control. The number of VCs per router per direction suggested by the authors is limited to 4, which limits the number of simultaneously supported connections. In section IV-C and table V we compare the hardware costs of artNoC [15] to our solution, showing better performance at half the area.

III. Motivation and Design Outline

A number of previous solutions rely on a BE NoC to support dynamic establishment of connections. For distributed dynamic solutions [5], [9], the BE NoC is not only used for delivering configuration packets, but also for path searching. Centralized dynamic solutions also depend on a BE NoC [8] for delivery of configuration information. Stefan et al. eliminate the BE NoC in dAElite but then they add a tree shaped dedicated configuration network instead [2]. In AElite certain time slots of a router are still reserved for configuration message delivery [6]. In addition, the configuration time of hundreds of cycles is very long.

An additional network for configuration is not cost-effective [1]. The to-be-allocated resources of the CS network have to be free at the time of path search and setup such that they can be used for the very task of connection setup. Besides, delivering all kinds of messages for connection establishment as BE packets is also inefficient and limits the selection of routing algorithm, as we mentioned in Section II.

To avoid these drawbacks, we propose a probe based solution for TDM NoC, where the probe searches through the network to find a free path, selects the time slots, and allocates the network resources for the required slots as it moves forward. When it arrives at the destination, the path is set up. As teardown, Ack/Nack messages are tightly combined with the probe, our solution is not restricted to deterministic path searching algorithms. We use the parallel probing algorithm in [14] but apply it to TDM NoCs. The probe uses the same wires and buffers as the data uses after the path is set up.

In the following description a *channel* denotes a simplex link between two routers together with associated buffers in a particular time slot; hence, the same link in different time slots belong to different channels. A probe is sent out by a source node and travels through free channels, moving from one router to the next towards the destination. If a free channel is available, the probe will reserve the channel for future data transfer and go through the channel to the next router to continue the search. If at some point no free channel is found, the path search fails. Ack/Nack messages must be sent back to inform the source node whether a search failed or succeeded. Besides, Nack signals also tear down the reserved channels of a connection. To reduce wire costs, backward Ack/Nack messages are 1-2 bits. In probe based setup methods both forward (data/probe) and backward (Ack/Nack) signals are needed.

Backward Ack/Nack messages constitute a design challenge, since they associate to a connection, consist of only 1-2 bits, contain no address information and share wires in TDM manner. Thus, they must arrive at the right router in the right time slot and rely on the slot table's information of the router to move back towards the source. In figure 1 a connection spans 5 routers. For the forward data/probe path, the reserved time slots inside each router follow the sequence $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$. The slot table of each router records the crossbar configuration information. To ensure that backward Ack/Nack signals correctly read the information inside each slot table for traversal, the backward signals should be sent-out at slot 0 in router 5, then reach router 4 at slot 3, reach router 3 at slot 2, and so forth.

To address this challenge we introduce a double orientation time wheel. Inside each router, there are two slot counters, of which one is incremented and the other is decremented; both start from slot 0. The incrementing slot counter uses the slot table to configure the forward crossbar for data or probe; the decrementing counter configures the backward crossbar for Ack/Nack. In this way, if the Ack/Nack signal is sent out at the correct time slot, it will be correctly routed back to the source hop by hop.

IV. Router Design and Implementation

A. Control signals

The implementation uses a mesh topology with 5-port routers where each port consists of two physical links in opposite direction. Each link contains a data path, which is used for delivering the probe during the setup phase and for data transmission after a connection has been established. Every data path is coupled with 4 control bits: a 2-bit Request signal in parallel to the data path, and a 2-bit answer (ANS) signal in the opposite direction of the data path. Their usages are listed in tables I and II, respectively. Ack/Nack messages are carried by the ANS signal. In the data transfer phase the same ANS signal can be used for end-to-end flow control, as shown in table II.

TABLE I
The usage of Request signal

Request	Usage
00	Idle/data transfer
01	Unused
10	Probe comes in
11	tear-down established connections

TABLE II
THE USAGE OF ANS SIGNAL

ANS	Usage in setup	Usage in data transfer
00	Idle	Ready to receive data
01	Unused	Unused
10	Nack (Path search failed)	Unused
11	Ack (Path established)	Receiver buffer full

TABLE III
THE PROBE FORMAT

Lookahead routing (5 bits)	Dest.addr (6 bits)	Src.addr (6 bits)

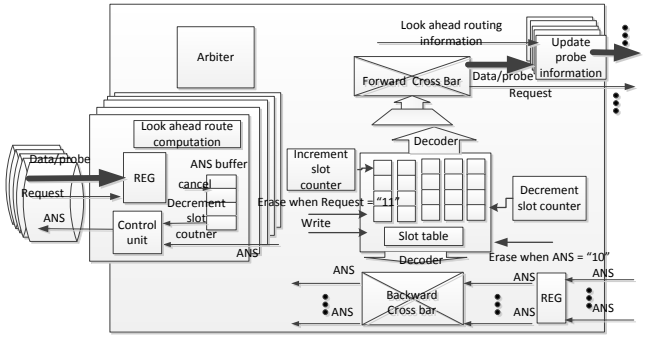


Fig. 3. Detailed router architecture

In total there are only 4 control wires for each channel, which, we believe, is the minimum overhead for probing based setup. The probe format is also compact. A probe just contains the destination address and look-ahead routing information. In an 8×8 mesh, the minimum width of a probe is 11 bits. Since the source node address is required by certain path searching algorithms (e.g. our parallel probing in this paper), a probe can become 17 bits in this case, as shown in table III.

B. Detailed router architecture

The slot-table structure is illustrated in figure 3. Rows in a slot-table represent time slots, and columns denote output links. The input channel id (In a 5 port router, it equals to $\lceil \log 2^5 \rceil = 3$ bits) is written into a vacant cell to book an output link for a certain slot. Each router has 5 outputs and thus the slot table width is 15 bits. This slot-table information, accessed by the incrementing slot counter, is used for forward crossbar configuration, while the decrementing slot counter is in charge of the backward direction. The cell content including its column number can be translated into two sets of configuration bits for the configuration of the forward crossbar and the backward crossbar, respectively. The translation logic is a simple logic decoder.

Inside each router, all input signals are latched. Then, the request signal is checked as follows:

"00" **data:** the data is directly forwarded according to the the corresponding slot table cell;

"11" **erase connection:** This signal will be firstly delivered through the crossbar according to the reserved slot table's information. Then, the corresponding slot table cell can be cleared at the beginning of the next cycle.

"10" **setup probe:** Based on the look-ahead routing information of the probe, arbitration for output channels commences. If a probe fails, backward ANS is used to notify the source node and cancel the reserved slot table cells hop by hop. We will explain this complicated process later. If a probe succeeds in acquiring one cell, it is delivered through the crossbar to the next router immediately. The slot table update is removed from the critical timing path and scheduled at the very beginning of the next cycle.

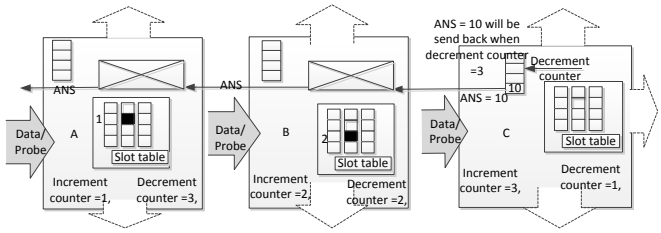


Fig. 4. ANS signal management

To reduce the critical timing path, look-ahead routing is used. The lookahead information denotes the desired output directions of the next router of a probe. This information is pre-computed in parallel with the arbitration and cross-bar traversal process of a probe. After the crossbar traversal, the probe will be updated and carry the new information before it reaches the next router.

The proposed router takes one slot per hop for all messages: request, probe/data as well as Ack/Nack. In our implementation, by default each time slot is one cycle.

Several aspects of our router deserve to be noticed.

1) Backward ANS signal management: We have mentioned in Section III that Ack/Nack signals must be sent out at the correct time slot. In the following we explain how to realize this scheme.

As figure 4 illustrates, a probe arrives when router C at incrementing slot counter is 3 and decrementing slot counter is 1. However, when it fails in router C, a signal ANS="10" will be sent back. The ANS should use slots $3 \rightarrow 2 \rightarrow 1$ to pass routers C, B, and A, respectively. To this end, ANS is buffered in router C for 2 slots, until the decrementing slot counter becomes 3.

We designed an ANS table at each input port for such buffering purpose. Each table cell is 2 bits wide. The rows represent time slots. The writing position of the table is pointed at by the incrementing slot counter; the reading position is pointed at by the decrementing slot counter. After a cell is read and ANS is sent back, it will be erased at the beginning of the next cycle. The maximum buffering time is K cycles, where K is the total slot number in the time window.

2) Predictable delay and setup polices: All kinds of message delays in our NoC are predictable, which are proportional to the distance between source and destination, with 1 cycle per hop (by default a slot corresponds to 1 cycle).

Moreover, the delay of a single search is also predictable. Suppose the distance between source and destination is D and the total slot number in the time window is K. Assume minimal routing. Then it takes at most $2D + K + 6$ cycles for the source to receive an Ack/Nack. D cycles are for sending the probe from source to destination, $D+K$ cycles for returning the ANS signal, and 6 cycles are consumed in the source and destination nodes.

We study two connection setup polices (adopted from [14]):

Retry until success: In this policy, the source node keeps retrying a request until it successfully sets up a connection. In this case the worst delay for setup is unbounded, because it is unknown when a free path becomes available.

Retry before deadline: In this policy, a deadline is attached to each setup request, which denotes the time when the connection has to be set up. The *residual time (RT)* is the deadline minus the current time. Since it can take $2D + K + 6$ cycles to set up a connection, a new connection is launched or an old is retried only when $RT > 2D + K + 6$. Otherwise, the request is discarded.

3) Path searching algorithms: Unlike other distributed path setup solutions [9], our probe based solution imposes no constraint on the routing algorithm. Hence, any probe searching

algorithm with reasonable hardware cost is applicable. However, to achieve high performance we implemented the adaptive parallel probing algorithm proposed by Liu et al. [14]. Parallel probing searches all shortest paths between a given source and destination in parallel. If at least one shortest path is free it will be found and allocated in constant time. To achieve this, a probe is copied from the input of a router to up to two outputs if there are two productive directions towards the destination. In the process many parallel paths may be allocated by the travelling probes, but all of them except one will be de-allocated as quickly as possible.

However, we replaced Liu's complicated priority comparison arbitration mechanism with a smaller and faster round-robin arbitration. Since each probe may have two productive output directions and may book two slot cells, the corresponding 2-bits ANS cell is used to record the number of booked slots. Hence, the value of a cell will be decreased when a Nack signal from the downstream router is sent back. Thus, it requires that a cell can be written by both the incrementing and decrementing counter. Fortunately, our router operating mechanism can guarantee that there is no conflict. A Nack signal is only returned to its upstream router when its own ANS cell value becomes zero.

4) Timing, synchronization and scalability considerations: Our design minimizes the critical timing path. The critical path length consists of an input probe checking logic (4 gate delays), a round-robin arbiter (8 gate delays), a multiplexer (2 gate delays), a crossbar (4 gate delays), and a look-ahead routing information updating logic (2 gate delays). Thus, our router is very fast even though it only contains 1 pipeline stage.

Our design can also be applied in a mesochronous or asynchronous environment by adding synchronization tokens for slot update handshaking, similar to the technique used in Æthereal [1]. Our double time-wheel design does not impose any additional requirement on synchronization. If such synchronization efforts are required, we need to add 1 pipeline stage in the router to compensate the latency introduced by synchronization.

Considering hardware scalability, the main cost of adding a time slot is the increase in buffers. For each router, this will raise 25-bit buffer space in total (15 bits for the slot table, 10 bits for the 5 ANS tables in total). In our current design, we use registers for storage.

C. Implementation costs and comparison

The router synthesis results with TSMC 90nm technology is listed in table III. In our default settings, the effective link width for data is 32 bits, and the total link width is 36 bits. The additional wires are always 4 bits. This value does not increase with network size or the number of slots[1].

TABLE IV
THE ROUTER SYNTHESIS RESULTS WITH 90NM TECHNOLOGY

Slots in the time window	Critical path length (ns)	Area (um^2)	Power (mW)
1	0.7	8730	5.8
4	0.7	12226	6.8
16	0.7	22608	9.7

As table IV suggests, the router area goes up linearly with the slot number. One additional slot increases the area by 991 um^2.

[1]Since the probe width is 17 bits, it requires that the forward data/probe path is at least 17-bit wide. However, if the data width used in data transfer is smaller than 17 bits, some wires of a link are inevitably wasted and thus regarded as additional wires.

We compare our synthesis results with other works reported in the literature. A conclusive comparison is difficult to perform because the different NoCs support different features. Nevertheless, we try to present a fair comparison based on available data.

The synthesis results of our router with TSMC 65nm, 90nm and 130nm, with different effective data path width and different slot numbers, are listed in table V and compared with others' results by assuming that all the routers are used for an 8×8 mesh. Generally speaking, our work has the shortest critical timing path[2]. Lusala's work [9] has the same critical timing path length but it is a mix of SDM channels and TDM channels. Its hardware cost is 5 times higher but offers more routing flexibility[3]. The area of our implementation is only slightly bigger than dAElite. However, dAElite [2] does not include the hardware cost for the scheduler. Finally, our work has the smallest additional wires per link for control/configuration[4].

TABLE V
COMPARISON WITH OTHER PUBLISHED DESIGNS BY USING THE SAME DATA PATH WIDTH AND SILICON TECHNOLOGY

	Critical path length (ns)	Area	Additional wires per link	Auxiliary network
Data width 16 bits, 130 nm technology				
artNoC [15] 2-flit buffers, 4 VCs	2	0.06mm^2	8	No
Our work 8-slot	1.3	0.03mm^2	5	No
Data width 48 bits, 65 nm technology				
Lusala [9] 3 SDM lanes, 3 time slots	0.5	0.05mm^2	20	Yes
Our work 9-slot	0.5	0.01mm^2	4	No
Data width 64 bits, 90 nm technology				
dAElite [2], 4 slots	above 1.08	0.016mm^2	10	Yes
Our work, 4-slot	0.7	0.017mm^2	4	No
Data width 68 bits, 130 nm technology				
HAGAR [8], 1 slot	2	20 kNand	5	Yes
Our work, 1-slot	1.3	5.5 kNand	4	No

V. PERFORMANCE EVALUATION

A. Experiment settings

Each resource node generates setup requests according to a Poisson distribution and pushes them into a queue. Uniform random, shuffle, and other traffic patterns are used for evaluation. An FSM pops a request from the queue and sends it out when an output slot is available. Then the FSM waits for a success or failure notification, upon which it either retries the request, discards it, or commences the data transfer. Any data point that is shown in the figures comes from simulation of 10 million cycles, of which the first 5% are discarded as a warm up period.

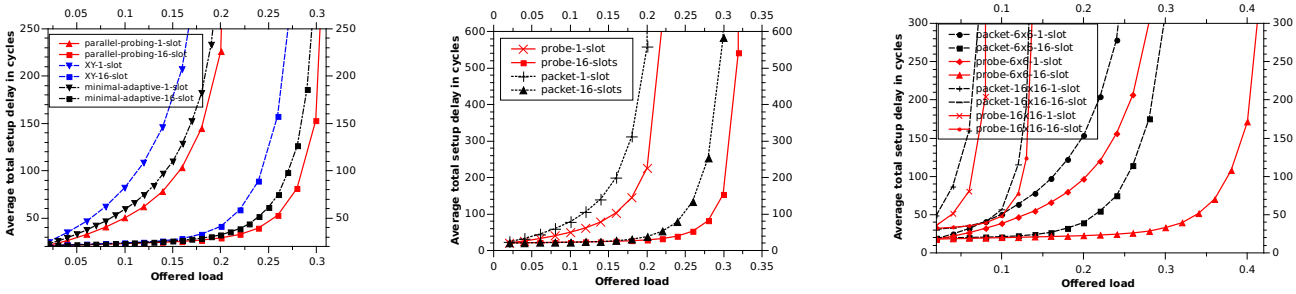
Several performance metrics are used:

- Metrics for retry until success policy:
  total setup delay includes the *waiting time* for a request in the queue and the *setup delay* for a request, which extends from the first time sending the request until final success.

[2]The dAElite [2] [18] did not report its critical path for a 5-port router with 90 nm. However, the critical timing path of a 3 port router with 65nm is reported as 1.08 ns. Thus it is safe to deduce that with 90 nm, it is more than 1.08ns.

[3]In SDM any of the 4 lanes of an input port can be forwarded to any of the 4 lanes of an output port. But in our TDM scheme, one TDM time-slot can be forwarded only to the next time slot.

[4]From the implementation of HAGAR, we deduce that each link requires 2 bits to connect to the center node, 1 bit to distinguish the BE and GS packets, 1 bit to distinguish normal GS flits and link free signal, and 1 bit for stall/go flow control, so the additional wires are 5 bits/link in total.



(a) Comparison of path searching algorithms.

(b) Comparison with packet based setup in 8×8 mesh.

(c) Comparison with packet based setup in 6×6 and 16×16 meshes.

Fig. 5. Performance comparison under uniform traffic. Each connection delivers 100 flits.

offered load refers to the required data transfer per connection multiplied by the injection rate of setup requests. Suppose the injection rate is 1/2000 cycles, and each connection delivers 100 flits of data after setup, then the offered load is 100/2000=0.05 flits/cycle.

- Metrics for retry before deadline policy:
  request success rate denotes the ratio between established and desired paths and indicates the number of the requested paths could be established.
  master percentage denotes the percentage of nodes which can send out setup requests. These nodes are uniformly randomly distributed in the system.

B. Comparison of different path searching algorithms

Three path searching algorithms are implemented and compared, which are X-Y, minimal adaptive and parallel probing. In this experiment, the retry until success policy is applied. After a connection is established, 100 flits of data are delivered before the connection is released. E.g. when the *window size* (total slot number in the time window) is 1 (no TDM link sharing), it takes 100 cycles for data delivery; when the window size is 16 slots, it takes 1600 cycles. The actual time for data delivery equals to the required data transfer time multiplied by window size.

The results under uniform random traffic are shown in figure 5a, suggesting that parallel probing is the best path searching algorithm. E.g. at offered load 0.16 and when the window size is 1, the average setup delay of parallel probing is only 80% of minimal adaptive, and 50% of the X-Y algorithm. We also have evaluated algorithms with different window sizes, e.g. 16 slots. Their results suggest the same ranking. Consequently we choose parallel probing as the default path searching algorithm for the following experiments.

C. Comparison with distributed setup

We re-implement a packet based distributed path setup method according to Lusala's work [9] for comparison. Since with a BE packet based setup method, the message delay is unpredictable, we apply the retry until success policy.

The average total setup delay versus offered load in an 8×8 mesh under uniform random traffic is shown in figure 5b. We observe that our parallel probing method has shorter average setup delay than the packet setup method of [9]. E.g. when the window size is 16, at load 0.26, the average total delay of our probing method (refer to probe-16-slot) is 52 cycles, while the packet based method needs 134 cycles. Also, the saturation point of the network is 15-18% higher in our probe based solution, which translates into a correspondingly higher network utilization.

We also observe that increasing the number of slots in a time window can help to ameliorate both network utilization and setup delay, although the time required for data delivery and hardware costs increases.

Besides 8×8 NoC, we made comparison in different network sizes (6×6 and 16×16) and with different slot number (1 and 16), see figure 5c. We use solid, red lines to represent the setup delay of our method and dashed, black lines for the packet based method. The probe based setup has 30% to 80% lower delay and a 10% to 40% higher saturation point in this comparison.

In addition to uniform random, we use other traffic patterns for evaluation. As figure 6 shows, under shuffle traffic probe based setup has 81% delay reduction and an up to 110% higher saturation point. Under tornado traffic, we find upto 50% delay reduction and a 55% higher saturation point (not shown in the figure).

Thus, we conclude that our method offers better performance than the packet based method and without any auxiliary network. The drawbacks of the packet based method discussed in section II accounts for its inferiority.

D. Comparison with centralized setup

In this section, we will compare with two different dynamic centralized setup solutions.

1) Comparison with a centralized solution using hardware accelerator for path scheduling: We compare with HAGAR [8] by choosing the retry for deadline policy. We use *request success rate* versus *offered load*[5] as metric, since it has been reported in [8]. However, in [8], setup delay data is not reported. Actually the success rate should be related to the setup delay to make it a useful metric. Moreover, the limitation of [8] is that the supported window size is only 1.

We compare a 6×6 and a 16×16 network with 200 flits of data for each connection. The deadline is 200 cycles (just for setup not including data communication), which means the total setup delay of a successful request should be smaller than 200 cycles. Otherwise the request fails.

Figure 7 shows that the success rate of our method is better than HAGAR's. For example, in a 6×6 NoC at master percentage 50% and at offered load between 0.6 and 1.0, with 1 slot window size, our solution offers about 34% higher success rate; with 16 slots in total, this figure rises to 100%. In a 16×16 NoC, our design has even more advantages. At an offered load between 0.6 and 1.0, our solution offers 170% higher success rate with

[5]The metric *offered load* in this paper is the same as *route rate* in [8].

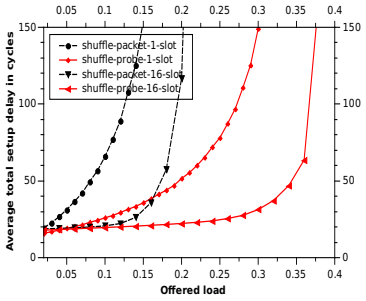


Fig. 6. Comparison with packet based setup in 8×8 mesh with shuffle traffic. Each connection delivers 100 flits.

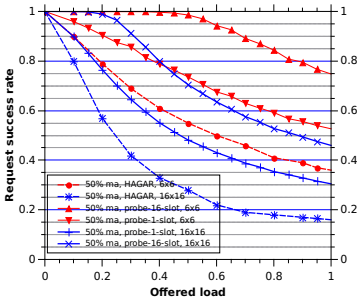


Fig. 7. Comparison with HARGAR [8] in 6×6 and 16×16 meshes. Each connection delivers 200 flits.

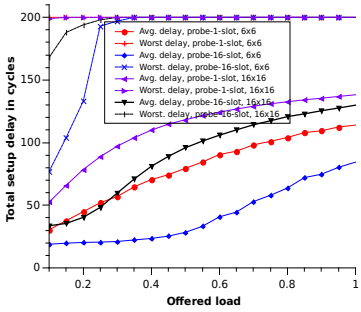


Fig. 8. Worst case and average delay of probe based setup with retry before deadline policy. Each connection delivers 200 flits.

16 slots in total. Again, increasing the window size enhances the setup performance.

The average and worst case total setup delay is reported in figure 8. The worst case delay (dashed lines) is always bounded and never exceeds 200 cycles, as required by the retry before deadline policy.

2) Comparison with a centralized solution using software based path scheduler: We compare with the software based centralized scheduler proposed by Stefan et al. [12], which can be used by dAElite [2]. Since the detailed delay analysis is not provided in that paper, we approximately calculate the expected delay figures based on the data given. The average time required for scheduling a path grows linearly with the distance between source and destination and for a distance of 4 hops it takes about 1200 cycles [12]. Thus, the sum of the request arrival rates of all the nodes together should not exceed 1/1200 per cycle, otherwise the scheduler is overloaded and requests have to be discarded.

Thus, let us ignore the path configuration and message communication delay and make an estimation. In a 6×6 network, with uniform random traffic, the average distance is 4 hops. Suppose each connection delivers 200 flits, then at offered load 0.1 the injection rate of requests per node per cycle is about 1/2000. At master percentage 50%, the total injection rate of all the nodes is $36/2000 * 0.5 = 9/1000$, which exceeds the capability of the scheduler. Even if all served requests can successfully find a path, the success rate is still less than $1/1200 \div 9/1000 \approx 9.3\%$, much less than in our work and HAGAR's (both above 90%). If path configuration and message communication delay is added, and considering the fact that not all served requests can succeed in finding a path, the success rate is even lower.

VI. Conclusion and Future Work

We have proposed a TDM circuit switching NoC with a parallel probing setup method by developing a double time-wheel technique. Our simulation results demonstrate that our solution is superior in terms of setup delay, network performance and hardware cost to all previously reported comparable solutions. The main reasons are due to

- that we use a distributed setup method that scales well with network size;
- that we propose a double time-wheel router with a parallel path search that searches effectively through all the possible shortest paths in parallel;
- that we use the available network resources also for path search, setup and configuration, thus incurring minimal extra hardware cost.

Although in our experiments each connection uses only one of the TDM slots of a time window, our design can support connections with multiple slots. This can be realized by sending out multiple requests for one connection, of which each request will build a path with one slot. However, this is inelegant and raises complications due to massive contentions. Therefore, we will tackle this problem next by developing a sophisticated technique for allocating multiple slots within a time window for a connection.

References

[1] K. Goossens and A. Hansson, "The Æthereal network on chip after ten years: Goals, evolution, lessons, and future," in *DAC*, 2010

[2] R. Stefan, A. Molnos, and K. Goossens, "dAElite: a TDM NoC supporting QoS, multicast, and fast connection set-up," *IEEE Transactions on Computers*, vol. PP, no. 99, p. 1, 2012.

[3] A. Banerjee, P. Wolkotte, R. Mullins, S. Moore, and G. J. M. Smit, "An energy and performance exploration of network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, no. 3, pp. 319–329, 2009.

[4] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *DATE*, 2004.

[5] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: concepts, architectures, and implementations," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[6] A. Hansson, M. Subburaman, and K. Goossens, "Aelite: A flit-synchronous network on chip with composable and predictable services," in *DATE*, 2009.

[7] J. W. Lee, M. C. Ng, and K. Asanovi, "Globally synchronized frames for guaranteed quality-of-service in on-chip networks," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1401–1411, 2012.

[8] M. Winter and G. Fettweis, "Guaranteed service virtual channel allocation in NoCs for run-time task scheduling," in *DATE*, 2011.

[9] A. K. Lusala and J.-D. Legat, "Combining SDM-Based circuit switching with packet switching in a router for on-chip networks," *International Journal of Reconfigurable Computing*, vol. 2012, pp. 1–16, 2012.

[10] P.-H. Pham, P. Mau, J. Kim, and C. Kim, "An on-chip network fabric supporting coarse-grained processor array," *IEEE Transactions on Very Large Scale Integration Systems*, vol. PP, no. 99, pp. 1 –5, 2012.

[11] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, and H. Corporaal, "Dynamic time-slot allocation for QoS enabled networks on chip," in *3rd Workshop on Embedded Systems for Real-Time Multimedia*, 2005.

[12] R. Stefan, A. B. Nejad, and K. Goossens, "Online allocation for contention-free-routing NoCs," in *ACM Proceedings of Interconnection Network Architecture: On-Chip, Multi-Chip Workshop*, 2012.

[13] D. Wiklund and D. Liu, "SoCbus: switched network on chip for hard real time embedded systems," in *IEEE Parallel and Distributed Processing Symposium*, 2003.

[14] S. Liu, A. Jantsch, and Z. Lu, "Parallel probing: Dynamic and constant time setup procedure in circuit switching NoC," in *DATE*,2012.

[15] C. Schuck, S. Lamparth, and J. Becker, "artNoC - a novel multi-functional router architecture for organic computing," in *FPL*, 2007.

[16] N. Kavaldjiev, G. J. Smit, P. G. Jansen, and P. T. Wolkotte, "A virtual channel network-on-chip for GT and BE traffic," in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, 2006.

[17] E. Mensink, D. Schinkel, E. A. Klumperink, E. van Tuijl, and B. Nauta, "Power efficient gigabit communication over capacitively driven rc-limited on-chip interconnects," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 2, pp. 447–457, 2010.

[18] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens, "A TDM NoC supporting QoS, multicast, and fast connection set-up," in *DATE*, 2012.