Parallel Probing: Dynamic and Constant Time Setup Procedure in Circuit Switching NoC

Shaoteng Liu, Axel Jantsch, Zhonghai Lu
Royal Institute of Technology, Sweden

***Abstract*- We propose a circuit switching Network-on-chip with a parallel probe searching setup method, which can search the entire network in constant time, only dependent on the network size but independent of the network load. Under a specific search policy, the setup procedure is guaranteed to terminate in time 3D+6 cycles, where D is the geometric distance between source and destination. If a path can be found, the method succeeds in 3D+6 cycles; if a path cannot be found, it fails in maximum 3D+6 cycles. Compared to previous work, our method can reduce the setup time and enhance the success rate of setups. Our experiments show that compared with a sequential probe searching method, this method can reduce the search time by up to 20%. Compared with a centralized channel allocator method, this method can enhance the success rate by up to 20%.**

1. INTRODUCTION AND RELATED WORK

Several NoCs offer guaranteed services to meet the QoS demand of applications [1][2][3][4][5]. For example, some of them utilize packet switching mechanism with time division multiplexing channels [2][4][5], while others utilize pure circuit switching mechanism [1][3]. They all adopt the idea of setting up a dedicated path before data can be transferred. A main challenge is how to efficiently search a path and allocate the communication resources for it.

Some methods try to solve this problem at compilation time [6][7]. However, as mentioned in [8], they face the difficulty that applications like H.264 [11] and the possibility of several applications running in parallel do not allow an efficient static policy with task mapping and channel allocation. Thus, dynamic path searching methods are a flexible alternative.

Winter and Fettweis [2][8] developed a *centralized* way to realize dynamic path searching and channel allocation. They designed a Network-on-chip Channel Allocator for the Aetherial NoC [5]. One of the nodes is designated as the "NoCmanager". Inside this node, a special component called Hardware Graph Array (HAGAR) is used. HAGAR stores all channel usage information of the network. If other nodes try to set up a guaranteed path, they must first send their requests to the NoCmanager node via a best effort network, which is the Aetherial packet switching network. Then the NoCmanager node starts to deal with the request and uses HAGAR to compute the path. After computation, the NoC manager will send back the routing information in order to set-up the guaranteed service path as requested. The NoC Channel Allocator is a centralized solution for dynamic path configuration. The advantage is that the NoCmanager node can work very fast. The disadvantage is that it is not a scalable solution. Also, since requests are sent via a best effort network to the "NoCmanager", the delay of setting up a path is neither predictable nor guaranteed.

To overcome such a scalability issue, distributed path searching method was developed. Pham et al. [1] designed a Backtracking Wave-pipeline circuit switching NoC, which supports sequential probe search. During the path searching phase, a probe is sent out. As the probe travels from the source node towards the destination, it reserves the channels it has passed for future data transfer. When this probe encounters congestions, it will backtrack one hop, cancel the last channel it has booked, and try another way. When this probe finally reaches the destination, a path is established and the data transfer phase can be launched. If no path can be found, eventually the probe will backtrack to the source node. This distributed method with circuit switching mechanism has the advantage of supporting many nodes searching their path in parallel. But the disadvantage is that when the majority of circuit links are already in use, the backtracking based search may take a long time.

In this paper, we develop a parallel probe searching approach. Our work has the following properties:

1) The parallel probing can be combined with several different retry policies, that lead to different trade-offs and properties.
2) Under no-retry policy, if a shortest path connection can be found, it is guaranteed to be found in exactly 3*D +6 cycles; if the search fails, it fails in maximum 3*D+6 cycles.
3) Under retry-for-free-path policy, if a free path exists, it will always be found in maximum $6N^{1.5}$ cycles, with N being the number of nodes.
4) On average our method has shorter setup latency than previous methods.
5) The switch has a simple structure with an inexpensive and efficient implementation.

We have simulated our design and compared it with above mentioned work of [1] and [2]. The results show that our work has advantages in delay, success rate and area.

2. DETAILS OF PARALLEL PROBING

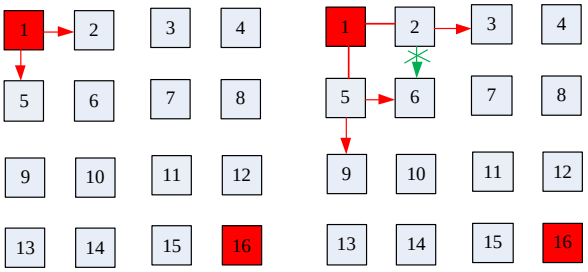
2.1. Intuition

As shown in Fig. 1, node 1 tries to set-up a path to the destination node 16. During the first cycle the source node sends out two probes to the neighboring nodes 2 and 5. In the second hop each probe splits into two probes and both continue to travel towards the destination along all possible minimum paths.

In the third hop node 6 receives two probes from the same setup request. One of them is cancelled and all the channels it has booked before are released. However, the channel between node 1 and node 2 is not released, because it is still needed for the probe that has travelled further to node 3. In this way a

978-3-9810801-8-6/DATE12/©2012 EDAA

wavefront of probes travel through the network and reach the destination on a minimal path. The time is exactly 3D, where D is the distance in terms of hops, and it takes 3 cycles to traverse each hop. When a probe successfully reaches the destination, an acknowledgement is sent back to the source node.



a) In each node a probe may double. b) When two probes meet, one is cancelled.

Fig. 1 An example of the parallel probe searching method.

Whenever two probes of the same request meet, one of them is regarded as redundant and is canceled, as shown in Fig. 1 b), and all channels used only by the canceled probe are released. The cancellation process proceeds backwards hop by hop. The switch does a cancellation based on the stored connectivity information that binds an input port to an output port. When a cancel signal appears on an output port from a downstream switch, the corresponding input port is looked up, the connection is canceled, and the cancel signal is forwarded upstream to the input port. Applying this mechanism, if several possible paths exist, one and only one of them can be finally booked, just as desired.

2.2. Structure of the switch

Fig. 2 shows the interface of each switch to the neighboring switches and to the local node. In a mesh topology, every switch is connected to its four neighbors and to the local resource node. Each link has a duplex data channel. This data channel is used for carrying the probe during setup and for transmitting data when a connection has been established. Each probe is one "flit" length. Every data channel is associated with an answer (ANS) signal consisting of 2 bit, which goes in the opposite direction to data channel, and 1 bit for a Request signal, which travels in the same direction as data channel. When the request signal is logic '1', a probe search is running or data transfer is active. When request signal is '0', it denotes idle state, and an established path will be released. The usage of ANS signal is listed in TABLE 1, which will be introduced in the following section.

TABLE 1 THE USAGE OF ANS (2 BITS)

Value	Usage
00	Path search continue/Destination is idle
01	Path cancel due to contention
10	Path cancel due to blockage
11	Path established/Busy destination

2.3. Operation flow

As shown in Fig. 3, our circuit switching network has six operation phases. The details are explained in the following.

2.3.1. Probe sendout

In this phase probes are generated and sent out. The request signal is set to '1'. The probe format is shown in TABLE 2, which contains source node address, destination node address, high level priority and low level priority. As a probe travels inside the network, it books the data channels together with the associate ANS and request signal. The probe itself is forwarded to the next node or nodes towards the destination. When a probe can move forward, the ANS to its previous node is set to "00".

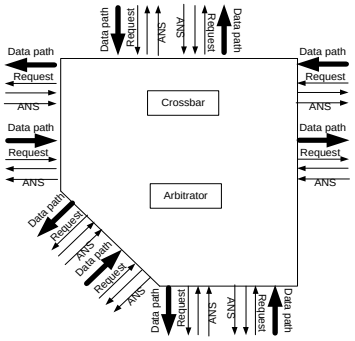


Fig. 2 Signal connection of a 2*2 mesh

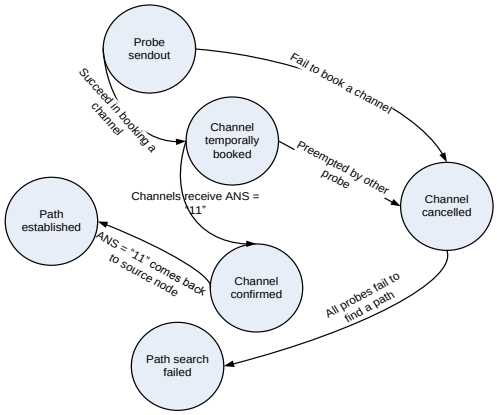


Fig. 3 Phases of operation

TABLE 2 Probe format

Src.Addr	Dest. Addr	High Priority	Low Priority

2.3.2. Channel temporally booked

A channel has 3 states, which are free, booked and confirmed. When a probe enters into a node, after winning arbitration, then

1)If the channels demanded by this probe are in free state, these channels are booked and switched into booked state.

2)If the channel demanded by this probe has already been booked by another probe, but this probe has a higher priority, it can preemptively book this channel. In this case, the path booked by the old probe will be cancelled.

If a probe succeeds in booking one channel, the ANS signal will remain "00".

2.3.3. Channel cancelled

Since a probe can have up to two desired channels in different directions when it enters a node, if and only if the probe is unable to book any channels, then we regard this probe as "failed". When a probe fails, the channel between the previous node and the current node is cancelled. Several factors can cause a probe to fail:

1) Its priority is lower than the probes which are demanding the same channel. This situation is caused by either the

probe loosing arbitration, or the channel booked by this probe is preempted by others.

2) Two probes carrying the same set-up request meet at the same node. One of them succeeds and the other fails.
3) All desired channels are used up by other connections and are in confirmed state.

If the failure is caused by the case 1), the ANS signal will be set to "01". If the failure is due to case 2) or 3), the ANS signal will be set to "10". Both "01" and "10" ANS will inform the previous node to release the channel.

Since a probe may have two desired directions, it is possible that a probe has case 1) failure in one direction, and case 2) or 3) failure in the other. In this situation case 1) always has higher priority.

2.3.4. Channel confirmed

When a probe reaches its destination, the ANS signal is set to "11" and transferred back. Channels along this probe's path will turn into "confirmed" state after receiving ANS "11". Confirmed channels can no longer be preempted.

2.3.5. Path established

Finally, when a source node receives a "11" ANS signal, then a connection is established and data transfer can commence.

2.3.6. Path search fail

When the source node receives a "01" or "10" ANS signal, the path search request has failed, and the reasons are distinguished as follows:

1) If the ANS is "01", it means that one of its probes has once contended with other active probes, and lost because of its low priority.
2) If the ANS is "10", it means that the probe has searched the entire network, but no minimum path is currently available.

Using this distinction, different policies can be applied to achieve different effects, e.g. see Fig. 11.

2.4. Detailed switch structure

According to the operation flow, the internal structure of a switch is shown in Fig. 4. It is divided into two parts: *control path* and *data path*. The *data path* transfers data through the configured *data path* crossbar. The *control path* is used to set up or tear-down a *data path*. The *control path* and *data path* share the same input and output wires.

In the *control path*, there are two crossbars, internal probe crossbar and control signal crossbar. Besides the crossbars, there is one arbiter, 5 input and 5 output controllers.

The probe crossbar is used to transfer a probe from one input to one output. The control signal crossbar is used to transfer requests and the ANS signal to the corresponding output.

The arbiter is used to solve contention between input probes and probes that already book a channel. The arbiter compares their priority and decides which probe wins.

Inside the input controller there is a channel monitor, a failure type monitor and an FSM. The channel monitor records the channels booked by the current probe. If the number of channels booked by the current probe becomes zero, then the probe is regarded as "failed". Whether the ANS signal transfers back a "01" or a "10" is decided by the failure type monitor. The failure type monitor remembers the cause of a failure, as described in section 2.3.3. The possible FSM states are *idle*, *prepare*, *booked*, *cancellation* and *fixed*. Its state transition graph is shown in Fig. 5 a). For example, when the ANS signal "11" is received, the input controller changes its state to "fixed" and transfers the "11" ANS signal to the previous node.

The output controller monitors and changes the states of the corresponding channel. An FSM is used inside the output controller; its states are shown in Fig. 5 b).

A probe needs two clock cycles to travel through the entire *control path.*

2.5. Priority strategy

As we mentioned above, contention between probes carrying different requests is a key point in this parallel probing method. Therefore, we have to wisely design our priority policy to solve contention.

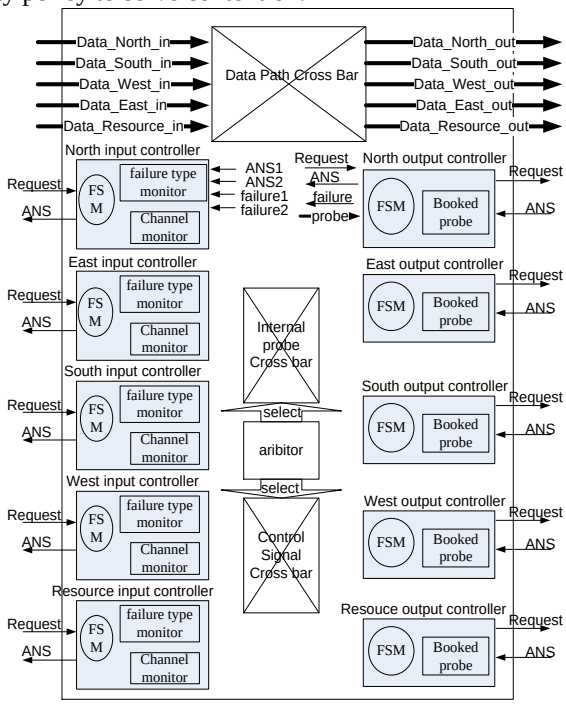


Fig. 4 Internal structure of a switch

We propose the following priority strategy:

1) The older the "age", the higher the priority. The "age" here can be understood as the time (in number of clock cycles) between current time and the first sent-out time of the request. The two level priorities are used to represent age.
2) Probes with higher priority can preempt channels booked by lower priority probes, if channels are in booked state.

This policy is used to avoid live lock cause by mutual blockage. Consider four requests A:1→3, B: 4→2, C: 2→4, D: 3→1 (Fig. 6). If these four requests are sent out at the same time, then they will block each other. Request A booked channel 1→4 and 1→2, then attempts to take channel 4→3 and 2→3. However, channel 4→3 has been booked by request B, and 2→3 by request C. Thus A is blocked by requests C and B. The situation is similar for requests B, C, and D. Without preemption, none of these probes can proceed, and retrying in a deterministic manner will not help. Thus, we use

preemption to ensure at least one of them can preempt the channels booked by the others and proceed to its destination.

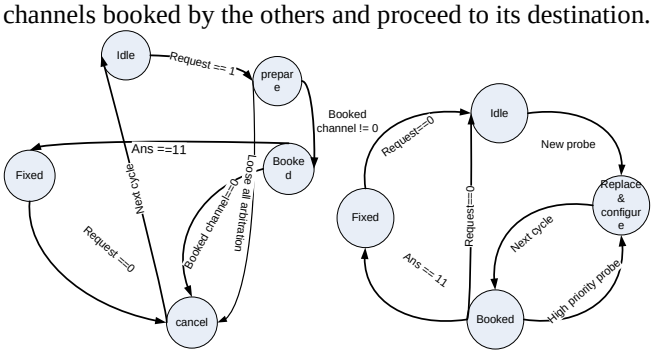


Fig. 5 a) FSM of input controller b) FSM of output controller

3) The source node id is used to avoid the stalemate when two different probes have the same priority. Rather than randomly selecting, the winner will be the one with the largest node id. This determinism leads to a winner-gets-all arbitration, which is required by retry-for-free-path policy (introduced in next section). Winner-gets-all arbitration is depicted in Fig. 6 b), suppose probe A and B with the same priority are contending for both channels 1 and 2. In winner-gets-all arbitration, one of them will win both channels, avoiding the situation that A gets channel 1 and B gets channel 2 which may for instance happen under random selection.

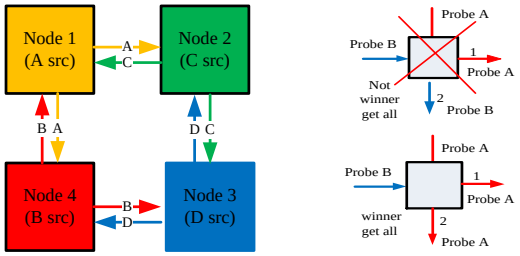


Fig. 6 a) live lock of probes **b)** Winner-gets-all

2.6. Time consumed in parallel probe search.

For every switch, it takes two cycles for a probe to traverse a switch, and it takes 1 cycle for the ANS signal to transfer back. So, it takes at most 3*D+6 cycles for a probe to travel from source to destination and send back the ANS signal (D is the hop distance between source and destination). 6 cycles is the overhead consumed in the source and destination nodes. Therefore, in an n*n mesh the worst case for a single search takes 3*(2*n-2)+6 cycles, no matter if the result is a success or a fail. In other words, it means the time for every single search is predictable and bounded, and has a complexity of O(n).

We have studied several policies.

1) **No-retry**. If a source node receives ANS "01" or "10", it will mark the request as "failed", then pick new request from the queue and send it out. Since every request just takes one single search, the maximum set-up time for a request in a n*n mesh is 3*(2*n-2)+6 cycles.

2) **Retry-for-free-path**. If the source node receives ANS "01", the probe failed due to contention with other active probes. This means there might be a free path but the probe failed to find it. After some delay, the source will retry the request until the ANS becomes "10" or "11", see Fig. 7Fig. 6. In experiments the retry interval is fixed to 3*(2*n-2)+6 cycles.

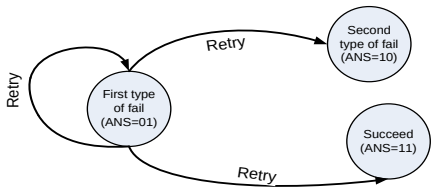


Fig. 7 Retry-for-free-path policy

The maximum number of retries required for a single search using retry-for-free-path policy can be calculated. Since the priority is increasing with the "age" of a probe, as the number of retries increases, the priority will also increase. Besides, one source node can only send out one request at a time, thus during every retry interval, there must be a node with the highest priority which never loses arbitration. Suppose at time t, $\alpha*n^2$ nodes are sending out requests, where α is the percentage of nodes which enabled to send out set-up requests (called *master percentage*). So the max retry times for a request to finish a retry-for-free-path search is $\alpha*n^2$, because during previous $\alpha*n^2$ -1 retry intervals, $\alpha*n^2$-1 other requests have finished their search, and this one has become the "oldest" one with the highest priority. It will finish a retry-for-free-path search without failure due to contention. In this case the set-up time spent for a single search using this policy is $\alpha*n^2$ *[3(2n-2)+6] cycles. And the time complexity is $O(n^3)$.

3) **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 search time is unbounded, because it is unknown when a free path becomes available.

3. EXPERIMENT AND SIMULATION RESULTS

3.1. Simulation method

As in Fig. 8, in each node a request generator generates set-up requests according to certain probability and pushes them into a queue. A FSM take a request out of the queue and send it out when the previous request has been accomplished or abandoned. After sending out a request, the FSM waits for the ANS signal to decide what to do next. In our experiment, we have studied the three kind of policies mentioned above.

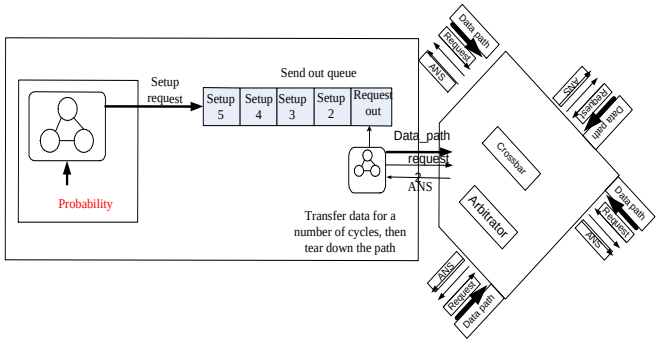


Fig. 8 experiment setup

3.2. Experiment result

In order to compare with the single probe searching technique with backtracking, we simulated an 8*8 mesh network. The simulation was performed under uniform random traffic of requests. The duration of data transmission (lifetime) was 400 probe cycles after the path had been established. The inter-arrival times of requests obey a Poisson

distribution. We used the retry-until-success policy. Each source sends out 3000 set-up requests, and first 300 and last 300 are discarded because of warm-up and tail phases. The *total delay* includes the waiting time for a request in the queue and the *setup delay* for a request, which extends from first time sending until the final success.

The average *total delay* versus offered load is shown in Fig. 9. The delay data for sequential probe is extracted from [1]. Here *offered load* refers to the duration of a path times injection rate. Suppose injection rate of requests is 1/2000 cycles, and the duration (lifetime) of a path is 400 cycles, then the offered load is 400/2000=0.2.

As shown in Fig. 9, our parallel probing outperforms sequential probe searching with backtracking. For example, the turning point in our case is delayed until offered load is 0.25, and the delay at 0.2 load is 21% reduced.

In order to compare with the centralized HAGAR solution [2], we use *request success rate* versus *route rate*. The *request success rate* denotes the ratio between established and desired paths and indicates how many of the requested paths could be established. *Route rate* refers to the portion of clock cycles in which a node is used for transferring data [2]. And *master percentage* means the percentage of nodes which can send out set-up requests. They are uniformly randomly distributed in the system.

$$route\ rate = \frac{paths\ wanted * lifte\ time}{simulation\ cycles}$$

We simulated 5,000,000 cycles, of which the first 1,000,000 cycles were discarded as warm up.

However, the success rate should be related to the setup delay to make it a useful metric. In [2][8] setup delay data is not reported. In Fig. 10, we use the retry-for-free-path policy to compare with HAGAR[2]. Our method has a better success rate when route rate is between 0.1 and 0.8. In the range 0.1-0.2 parallel probing has a 20% higher success rate.

We also compared a 6*6 network with 200 cycle lifetime. In this case, parallel probing outperforms HAGAR at every point. Our method has around 50% improvement over HAGAR in terms of success rate at route rates 0.8-1.0. Also, in a 16*16 network with 1000 cycles lifetime parallel probing is superior by 50% for route rates 0.8-1.0. Due to page limitation, figures are not listed here.

In addition, we compared the three mentioned policies of our parallel probe searching method. Here we define *send-out success rate* as the ratio between succeeded requests and the requests sent out from the queue. It indicates the success probability of a single request after sending it out. The relationships between route rate and *send-out success rate* and delay are shown in Fig. 11 and Fig. 12, respectively. Fig. 12 shows the delays only up to the saturation point. We find that, 1) retry-until-success policy has a 100% *send-out success rate*, at the expense of long latency at high network loads. Even in a saturated network every request eventually gets served, but the delay is unbounded. 2) The *send-out success rate* of retry-for-free-path policy stays around 0.54 when route rate is greater than 0.3. This is because the maximum speed of sending out requests becomes less than the speed of generating requests. The average delay of a request waiting in the queue keeps increasing. Although more requests are generated, a limitation exits for the requests that can be sent out during a fixed time interval. Therefore, the *send-out success rate* stabilizes even if the route rate still grows. 3) No-retry policy has the worst *send-out success rate* but the best delay performance. In our experiments the requests generation rate never exceeds the requests are sent out rate, even as route rate reaches 1.0.

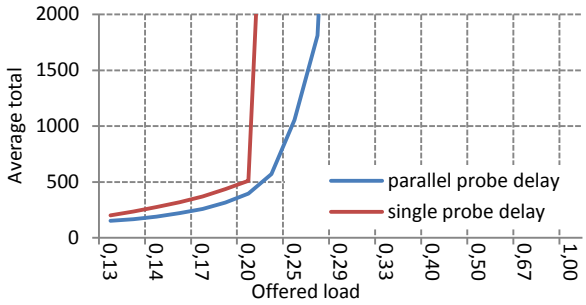


Fig. 9 Path set-up latency performance (8*8 mesh, lifetime 400)

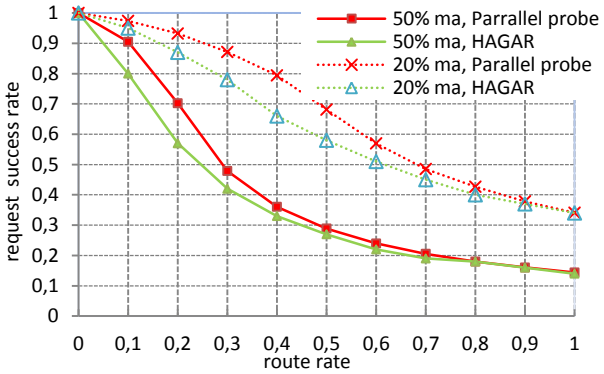


Fig. 10 Comparison of the different Path searching method, for lifetime 200 cycles at 16*16 mesh and master percentage 20% and %50

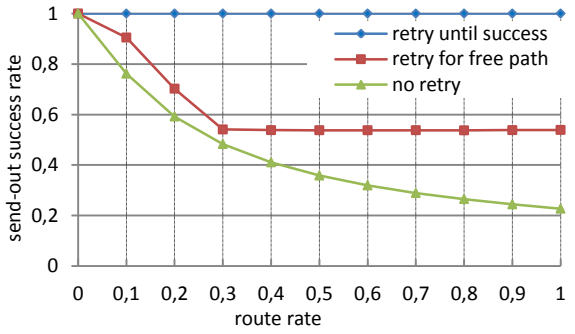


Fig. 11 success rate of the 3 policies of parallel probe searching method for a lifetime of 200 cycles with a 16*16 mesh and master percentage 50%

As mentioned, the *total delay* of a request is composed of delay of waiting in the queue and the *setup delay*, which extends from the first the time a request is sent out until the time the request is completed.

For retry-until-success policy, all delay types increase with the route rate without upper bounds. Some part of the average *total delay* and worst *total delay* curve is not shown in Fig. 12 when the delay was unbounded at that point. The delay in the queue keeps increasing and dominates at route rates above 0.2.

For the retry-for-free-path policy the average total delay and worst total delay also goes up with the route rates, and become unbounded above a route rate of 0.3. But the worst

setup delay is bounded, which is $\alpha*n^2*[3(2n-2)+6]= \alpha * 6N^{1.5}$ with $N=n^2$ being the number of nodes. For a 16*16 mesh with α (*master percentage*) %50, it is 12288 cycles. As shown in Fig. 12, the worst setup delay observed in our simulations is 2200 cycles. According to our experience, the theoretical worst case has a very low probability to occur.

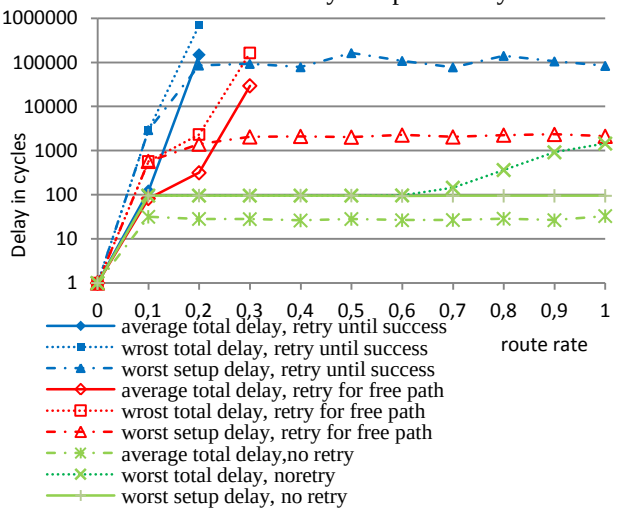


Fig. 12 Delay of the 3 policies of parallel probe searching method for lifetime 200 cycles at 16*16 mesh and master percentage 50%.

For the no-retry policy, the average total delay is around 36 cycles, and goes up slowly with the route rate. The worst case *setup delay* is also bounded. In theory it is 3*(2*n-2)+6=6n=96 cycles in a 16*16 mesh. In our simulations the worst case observed is exactly 96 cycles.

3.3. Synthesis results

Our switch has been synthesized by using Synopsys Design compiler with SMIC 90 nm library. The maximum clock frequency (probe clock) for the *control path* is 570MHz, the maximum clock frequency for *data path* is 1.8 GHz. This dual-clocking scheme has been well described and implemented in the work of Pham et al. [1]. It means that the switch operates at most at 570MHz during probe search stage. When the path has been established, it can use 1.8 GHz clock frequency to transfer source synchronized data [1]. In comparison, the sequential probe of [1] uses 0.18 um process and can work at 345MHz *control path* frequency and 923 MHz *data path* frequency. HAGAR [2] has been synthesized with FARADAY's 130nm UMC library and work at 200 MHz in an 8*8 mesh and at 50 MHz in a 16*16 mesh.

The area consumption for each switch node is 18733 NAND gates for a data path width of 64bits. Of that the *control path* consumes 10,364 NAND gates, and the *data path* consumes 8369 NAND gates. Hence, the per-bit area is 18733/64=292 gates. In comparison the sequential probe switch of [2] uses 12460/16=778 gates per bit.

Compared with the centralized solution with HAGAR, which has a payload of 68bit (32bit for address, 32 bit for data. 4bit for read/write/ mode), our solution is also better in terms of area, see Fig. 13.

4. CONCLUSION

We have proposed a circuit switched NoC with parallel probing, a parallel method for connection setup. Our simulation results demonstrate improvements in terms of setup delay, and success rate compared to previous work at comparable or reduced area.

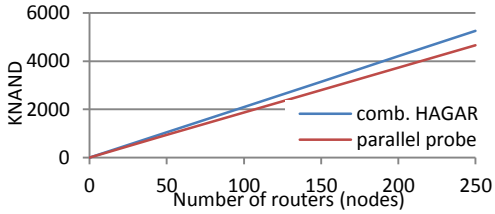


Fig. 13 Area consumption of all combined switches.

The special property of our switch is that the path search results can be acknowledged within a predictable, very low time limit under certain policies like retry-for-free-path or no-retry. In other words, worst case delay can be bounded in those policies. This property is important for real-time based applications. Hence, we have shown that parallel probing is an efficient and cost effective set-up procedure for circuit switched NoCs that can be used for dynamic circuit configuration in real-time and high performance applications.

However, due to the relative long setup time and the high resource usage during setup, it is only suitable for certain applications. For example, when life time of a path is short, a packet switching network will outperform our circuit switching. In future, we will do a comparison to identify the suitable application domain for circuit switched networks.

Furthermore, we plan to expand the flexibility of our method. . Currently entire links are reserved for a connection even if only a fraction of the link bandwidth is required. In future work we will consider the support for multiple sub-networks, which allows a connection to use only a fraction of a link.

5. REFERENCE

[1] P.-H. Pham, J. Park, P. Mau, C. Kim."Design and Implementation of Backtracking Wave-Pipeline Switch to Support Guaranteed Throughput in Network-on-Chip." *IEEE Trans. VLSI*, vol. 99, 2010.

[2] M. Winter and G.P. Fettweis "Guaranteed service virtual channel allocation in NoCs for run-time task scheduling." Design, Automation & Test in Europe Conference & Exhibition (DATE), Page 1-6, March 2011.

[3] D. Wiklund and L. Dake, "SoCBUS: Switched network on chip for hard real time embedded systems." In Proc. Int. Parallel Distrib. Process.Symp., 2003, p. 8.

[4] M. Millberg et al. "Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip." In Proc. of DATE, pages 890–895, February 2004.

[5] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: Concepts, architectures, and implementations." IEEE Des. Test. Comput., vol. 22, no. 5, pp. 414–421, 2005.

[6] A. Hansson, K. Goossens, and A. Radulescu. "A Unified Approach to Constrained Mapping and Routing on Network-on-Chip Architectures". In Proc. of 3rd Int. Conf. on HW/SW Codesign and System Synthesis, pages 75–80, 2005.

[7] J. Hu and R. Marculescu. "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints." In Proc. of DATE, pages 234–239, February 2004.

[8] M. Winter and G. Fettweis. "A Network-on-Chip Channel Allocator for Run-Time Task Scheduling in Multi-Processor System-on-Chips." In Proc. of 11th Euromicro Conference on Digital System Design (DSD), pages 133–140, September 2008.

[9] N. Ma, Z. Lu, L. Zheng "System design of full HD MVC decoding on mesh-based multicore NoCs." Journal Microprocessors & Microsystems Volume 35 Issue 2, March, 2011