Microprocessors and Microsystems 35 (2011) 130–138

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A TDM slot allocation flow based on multipath routing in NoCs

R. Stefan [a,*], K. Goossens [b]

[a] Technical University of Delft, The Netherlands
[b] Technical University of Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Available online 29 September 2010

Keywords:
Networks-on-chip
TDM
Multipath

ABSTRACT

Networks-on-chip have evolved as the natural solution for a scalable interconnect that can be automatically generated to suit the needs of the desired application. In this study we focus on improving the efficiency of on-chip networks using alternative routing strategies. We focus on a multi-path slot allocation method in networks with static resource reservations, in particular TDM NoCs. The simplicity of these networks makes it possible to implement this routing scheme without significant hardware overhead. Our proposed method, although displaying large variations between test cases, provides significant overall gains in terms of increased bandwidth or reduced working frequency or area. Our tests show that when using multipath routing the same communication requirements can be mapped on networks working on average at frequencies lower by 24.55% on average, while in individual cases the largest reduction was 60.04%. At the same time we are avoiding problems like deadlock and out-of-order delivery, commonly associated with multipath routing.

1. Introduction

Designing an efficient on-chip interconnect for systems-on-chip presents the engineer with multiple challenges. In addition to the raw performance requirements in terms of bandwidth and latency, the system designer has to consider possible interference between applications, the effect of crossing clock domains and guarantees regarding real-time system behavior. On the other hand the hardware requirements and power consumption of the interconnect have to be maintained at the minimum possible levels.

Networks-on-chip or NoCs represent the emerging paradigm for a scalable chip interconnect [1,2]. Among these, Circuit-switching NoCs [3–5] are an attractive solution as they are able to both isolate tasks from interfering with each other and they can provide communication channels with guaranteed bandwidth and latency. The time-division-multiplexing (TDM) technique employed in combination with circuit switching offers the means to divide the link bandwidth between channels with fine granularity and with discretionary budgets allocated to each channel.

We base our experiments on the Æthereal network [3,6]. Æthereal uses overall knowledge about the system behavior at design time to dimension the network and create allocations for each channel for each of the desired use cases. The allocation itself is performed by automatic tools which resemble in function the circuit routing tools, the main difference being that time is used as a degree of freedom in addition to space.

In the Æthereal implementation, the bandwidth of each link is split, in the time domain, into discrete allocation units called time slots. Typically, the entire bandwidth of each communication channel is allocated in one or more time slots along a single physical path. There may be the case though that no path is found that can satisfy the communication requirements. The designer is then forced to increase the hardware resources allocated to the network or its working frequency. As an alternative, it may be possible to divide the traffic of the channel that could not be handled using a single-path allocation over multiple physical paths.

In this study we evaluate the performance of an allocation flow which makes use of multi-path allocation in addition to the typical single-path algorithm. Communication channels are allocated one-by-one as in the traditional approach, but whenever the single-path search fails an allocation over multiple paths is attempted. This approach is shown to produce on average reductions in the working frequency of the NoC of 24.55% or alternatively can be used to reduce the size of the network necessary to support the communication requirements.

The rest of the paper is organized as follows. In the following section we present related work, and usage of multipath routing in other domains than networks-on-chip. Section 3 illustrates the hardware changes necessary to support our proposal. The TDM slot allocation algorithms are presented in Section 4. Experimental results are presented in Section 5 while the last section presents our conclusions.

* Corresponding author.
E-mail addresses: R.A.Stefan@tudelft.nl (R. Stefan), K.G.W.Goossens@tue.nl (K. Goossens).

0141-9331/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2010.09.007

2. Related work

In large scale networks, multipath routing has already been in use for a long time, for example in Internet traffic engineering [7] The problem presents other challenges though than the small-scale networks found on-chip. Buffering is in general plentiful by comparison and in-order delivery is not a requirement, because the protocol stack implements reordering in another layer, before the data is delivered to the user.

The problem of multipath routing in networks with resource reservation ie. asynchronous transfer mode or ATM was studied by Cidon et al. [8,9], and was shown to provide a benefit in terms of connection establishing time, while having mixed results from the bandwidth point of view.

Multipath routing in NoCs has been previously proposed in [10], however, the method presented there requires a complex mechanism for ensuring in-order delivery. Sequence identifiers are assigned to the packets and an arbiter rearranges them in the proper order at the point where the different paths converge. In contrast, in our solution the entire schedule of packets in-flight is known at design time and in-order delivery can be ensured entirely by selecting proper insertion time for each packet.

Multipath routing is also found in the various forms of adaptive routing or other forms of non-deterministic routing [11], largely addressed by studies of multiprocessors and now also applied to networks-on-chip [12–16]. The target of these studies is mainly guaranteeing deadlock freedom while maximizing utilization, but without explicitly addressing the costs of in-order delivery.

Our study targets NoCs that support resource reservation using time-division-multiplexing, in particular the Æthereal network [3]. Our algorithm performs both routing and slot allocation. Routing and slot allocation in a similar TDM setup is also studied in [17,18].

The same technique can also be applied to other TDM networks described in the literature, like the Nostrum network [5], aSoC [19] and the TDM test delivery in [20], the more flexible TDM technique of [21] and perhaps also to networks using space division circuit switching like [22].

A solution for performing mapping, single path routing and slot allocation in the Æthereal networks is presented in [23,24] performs in addition topology selection but using another network model which does not require slot allocation. The authors of [25] propose a graph coloring algorithm to solve a slot allocation problem in a similar network implementation but with more relaxed timing constraints.

3. Hardware architecture

The Æthereal network-on-chip implementation that we use in this study employs a routing model called contention-free routing [3,26]. Under this model, arbitration is avoided at the router level by ensuring at design time that packets only travel through the network according to a fixed schedule, which does not allow conflicts. The usage of each link is divided into fixed size time slots and each connection receives exclusive use of a subset of these time slots.

For proper functioning of the TDM schedule it is necessary that network elements are synchronized with their neighbors at flit level and agree on what is the current flit position within the schedule. In a typical synchronous implementation, a flit is composed of three words and thus transmitted during three clock cycles. One cycle would be spent for crossbar traversal, one on link traversal and one in the input buffers of the routers. However, it is not necessary that the network is synchronous at clock cycle level, even between direct neighbors. While the routing decision and crossbar traversal has to be simultaneous for all inputs of the router, asynchronous input buffers can be used and the time spent in buffers can vary to compensate for clock skew and link traversal time, as illustrated in Fig. 1.

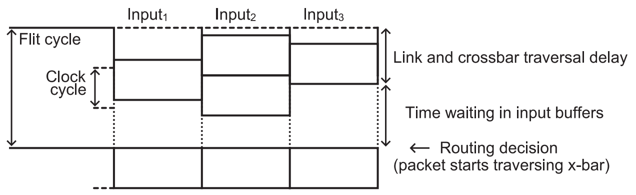

Input$_1$
Input$_2$
Input$_3$
Flit cycle
Clock cycle
Link and crossbar traversal delay
Time waiting in input buffers
← Routing decision (packet starts traversing x-bar)

Fig. 1. Flit-level synchronization.

Two main approaches are possible for implementing routing in compliance with this schedule. One consists of storing the routing information in a distributed manner in all routers and network interfaces (NIs) and the other employs source routing for deciding the path each packet should take. In both cases the NI is responsible for injecting packets into the network only in the allowed time slots.

3.1. Distributed routing

In the distributed routing implementation [27] each router is synchronized to a global clock at flit level and contains a routing schedule which specifies to which of the outputs each input has to be routed during each time slot. Once the slot tables are configured into the routers, the destination of each packet is determined solely by its time of insertion into the network.

Under this implementation, no modifications to the hardware are necessary in order to support multipath routing. Programmed with the right schedules, the routers can lead the packets on a variety of paths to destination. There is no cost related to the number of different paths used by a single connection and there is an additional benefit in that no headers are necessary for routing the packets, thus increasing the size of the useful payload.

The absence of routing headers reduces the cost of the multipath approach, however in our tests we have conservatively assumed the presence of these headers.

3.2. Source routing

The entire communication is performed in a predetermined manner, usually computed at design time. At run-time the path taken by packets and the time when one connection is allowed to use the physical links are read from tables inside the network interface.

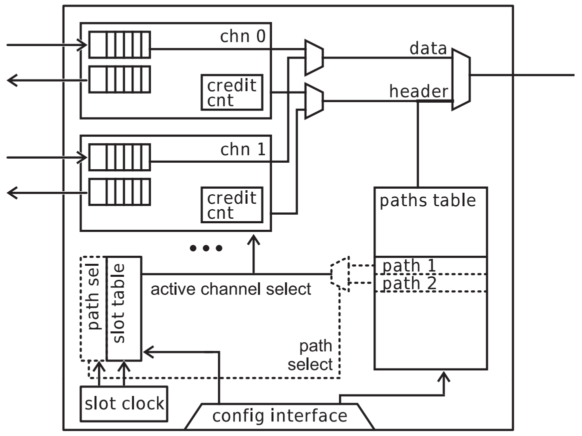


Fig. 2. Architecture of a network interface, the slot table dictates which channel is allowed to transmit at any given time; an additional table of paths selects the path to be used by the currently active channel.

This pre-determined behavior allows the network elements to run at high frequencies because the data can be read in advance as needed. The additional hardware required by multipath consists of enlarged tables for storing the routes and selecting the proper route at the proper moment in time (Fig. 2).

The changes in Fig. 2 have been implemented in hardware, and synthesized using Synopsys tools in 65 nm technology. We have found the overhead to be of 7.6% in terms of area for a network interface kernel with four channels and 16-word buffers, when four distinct paths are supported for each channel.

We have tested our design in FPGA in a setup with a MicroBlaze processor communicating over a 2 $\times$ 2 mesh network with a memory placed in the opposite corner of the network. Two distinct paths were setup for communication and we have verified in simulation that indeed both paths were used.

3.3. Freedom from deadlock and in-order delivery considerations

Two common problems associated with adaptive/multipath routing are the out-of-order delivery of packets and deadlock. Out-of-order delivery can arise when some packets take longer paths than others or are queued behind packets belonging to other streams in contention situations (Fig. 3a). Solutions to this problem consist of the reordering of packets at some point of convergence within the network or at destination, however this can result in the even more dreadful problem of network deadlock. If packets belonging to two different communication streams A and B, arrive out of order on two links, in such a manner that the first packet of A is queued behind the second packet of B and vice-versa (Fig. 3b), the router or NI producing the reordering will block, unable to service any of them. Even worse, because the size of buffers in networks-on-chip is always small, packets A_1 and B_1 might not even arrive in the queues of the element in charge of reordering, but instead they may be blocked somewhere upstream. This situation is similar to reassembly deadlock discussed in [28].

Our implementation avoids both of the mentioned problem. Because the employed mechanism is circuit switching and no arbitration takes place within the network deadlock cannot arise. Furthermore, even the set-up phase used to reserve the channels employs dedicated, circuit-switched connections and deadlock cannot arise even in the set-up phase. The lack of contention inside the network means the time taken by the packets for traversing the network is known beforehand, and even though some of them may take longer paths a schedule can be computed in such a way that "fast" packets never overtake "slow" ones. The algorithms for computing this schedule are presented in the following section.

4. Slot allocation algorithms

The problem of finding a slot allocation for a set of connections is similar to the problem of routing physical wires in integrated circuits. Compared to the wire routing problem, the number of connections our algorithm has to handle is relatively low, in the range of hundreds compared to tens of thousands of wires in digital IC design, but further complications arise from the fact that connections do not have equal bandwidth and the resources that need to be allocated to each connection vary.

Because this computation is expensive, it is generally performed at design time. Although we are currently investigating run-time allocation, which was independently demonstrated by [29,30], we consider it beyond the scope of this study. We would like to emphasize though that by performing the allocation at design time has the advantage that the run-time system behavior becomes completely predictable and can be easily modeled and verified.

The solution to the routing problem consists of successively allocating channels using a simple path-finding technique, while at the same time avoiding conflicts with other previously allocated channels. A possible extension consists of tearing down previously allocated channels obstructing the current path and reallocating them later. This approach is used by tools performing physical routing of wires, however in the current study we have not investigated this technique, instead focusing on improvements in the allocation of individual channels.

We perform our tests using three allocation algorithms. The first one attempts to allocate the entire communication bandwidth of each channel on a single path between source and destination, the second consists of iteratively applying the first technique and increasing the allowed path length until the bandwidth constrained is satisfied, while the third uses a flow algorithm to determine the slot allocation.

4.1. Single path

The technique used is the exhaustive search using a depth-limited backtracking. We first perform a breadth-first search using the Dijkstra algorithm to determine the shortest path from all nodes to destination.

The search considers the available bandwidth of each link but does not yet consider the alignment of the available slots. This step provides a bound on the minimum number of hops necessary for reaching the destination and also offers an early warning for cases when a solution does not exist at all. The real benefit of this search consists of the fact that it allows us to exclude solutions early during the depth-first exploration.

Consider the example graph in Fig. 4 representing a network topology on top of which several communication channels have already been allocated. Consider the problem of finding a path from the local NI of Router 20 to the local NI of Router 03. Furthermore consider that links R01–R02 and R21–R22 are already loaded to the extent that the bandwidth of the current communication channel cannot be satisfied regardless of the slot alignment. The distances to the destination NI, found by the breadth-first search are shown on the bottom-left side of each router.

The backtracking search starts at the source NI with a limit on the path length of 7. The backtracking algorithm constructs all possible paths leading to the destination computing at each node that is reached the subset of slots that can be used on the given path. The sum of the length of the path already traversed and the minimum distance to the destination as computed by the breadth-first search is not allowed to exceed the path length limit. This way the algorithm is forced to examine paths with a lower number of hops first.

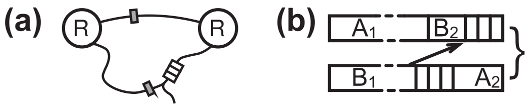


Fig. 3. Different delay causing out-of-order delivery (a) and deadlock situation (b).

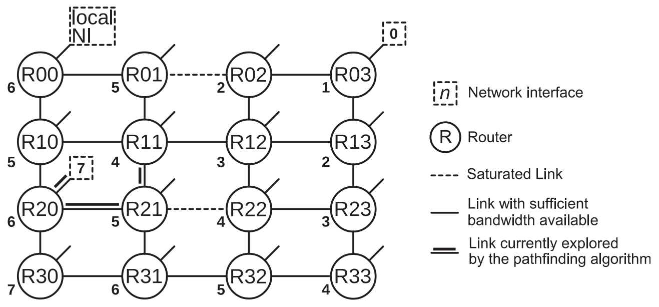


Fig. 4. Pathfinding example.

If a solution is not found, the path length limit is iteratively increased up to 16 more than the initial distance or until more that 10 million partial paths have been analyzed. These limits are necessary because the complexity of the backtracking algorithm grows exponentially with the number of steps and the downside is that the solution space searched is somewhat restricted.

The algorithm can be easily extended with further requirements, like restricting turns, allowing or disallowing the return to previously visited nodes or different cost functions for the traversal of different links.

4.2. Iterative maximum bandwidth search multipath

When only part of the desired bandwidth could be allocated for a communication channel we may attempt to allocate the remaining bandwidth over one ore more different paths using the same pathfinding algorithm. Slots already used are masked so that conflicts do not arise on links which are common to several paths, and guard slots are inserted before allocated slots when the maximum search path length is increased in order to guarantee that packets using shorter paths will not overtake the ones using longer paths.

The insertion of guard slots prevents the link from being used by the current communication channel, but it does not consume link bandwidth, in the sense that the same slot can be used by other channels instead.

In the example in Fig. 5 a packet sent in slot 2 will travel on a path with two intermediate hops and arrive in slot 5. If another path of length 4 is subsequently found, allowing a packet to depart in slot 1 would result in out-of-order arrivals, thus, a number of slots equal to the difference in path length also needs to be masked, although they are not in use on any of the traversed links.

Another tradeoff to be made is between the allocation of a larger number of shorter paths or of fewer longer paths. In this study we always give preference to shorter paths. The exact steps taken are presented in Algorithm 1.

Algorithm 1. Iterative Maximum Bandwidth search algorithm

```
maxPathLen = length of shortest path from source to
  destination;
set path search limit (maxPathLen);
while More bandwidth to be allocated do
  Attempt to allocate all remaining bandwidth;
  if succeeded then
    finish;
  end
  Attempt to allocate any bandwidth;
  if succeeded then
    subtract allocated bandwidth from desired bandwidth;
    mask allocated slots;
    continue;
  end
  if maximum number of paths reached then
    fail;
  maxPathLen = maxPathLen + 1;
  set path search limit (maxPathLen);
  if path length limit exceeded then
    fail;
  end
  add guard slots to mask;
end
```

To reduce the number of paths used, a single-path allocation is first run separately and the result is compared with the result of the multi-path allocation. The multi-path solution is used when the single path search fails, or when produces a better result. We consider the multi-path flow better if the total number of allocated slots over all edges is lower than the number of slots produced by the single-path search. Over all tests, we have found that the multi-path solution is used in only 1.67% of the cases, of which only one quarter was caused by a failure in the single-path allocation. This is important, because it implies that the hardware cost can be relatively low as the paths tables need to contain entries only for the paths actually used. Furthermore, in 81% of the cases only two paths were used, with three paths used in 16% of the cases and 4 in less than 3%.

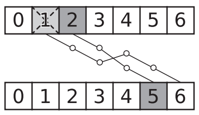


Fig. 5. In-order delivery is ensured through the use of guard-slots.

4.3. Flow algorithm

While for the previously presented algorithms physical edges on a path represent physical links, the flow algorithm uses individual slots of each physical link as the basic unit of allocation. All nodes and links in the original topology graph are split into a number of sub-nodes and links equal to the number of time slots as represented in Fig. 6 and are allocated independently.

The Edmonds–Karp flow algorithm [31] is used to allocate paths of one-slot bandwidth at a time, always having the minimum length allowed by previous allocations. Compared to sequential allocations over several paths, the flow algorithm has the advantage that it can displace previously allocated paths to make space for new ones. We also make use of heuristics to reduce the number of different paths used. For more details the reader is referred to [32].

The in-order delivery is verified after each allocation, and if necessary paths that would produce out-of-order messages are discarded. As with the iterative approach, if a single path solution exists and has lower cost in terms of allocated slots, it is used instead. On average, the solution produced by the flow algorithm is used in 3.61% of cases.

4.3.1. Path selection for in-order delivery

When a conflict exists between paths generated by the flow algorithm in the sense that they would produce out-of-order arrivals, we choose to discard some of the paths so that the remaining ones would only produce in-order deliveries. For selecting which paths should be discarded and which should not be, we use a deterministic algorithm based on dynamic programming.

The problem can be formulated as a Monotonic Subsequence Problem [33], for which optimal solutions exist with polynomial time complexity. The paths are ordered by slot departure time and the solution must comprise of a subsequence with only increasing arrival times.

Further complications arise from particularities of our problem. Because consecutive slots have a different payload efficiency (consecutive slots do not need to repeat the packet header), the items in the sequence need to be weighted, and, the algorithm needs to take into account the wrap-around that occurs at the end of the slot table.

The associated weight for each path does not introduce significant changes to the algorithm, but in order to cope with the wrap-around, the algorithm will have to be applied repeatedly in a window which slides over the list of paths. In some respects the wrap-around problem is similar to the one described in [34].

A formal description of the algorithm is given in Algorithm 2. The algorithm is optimal in the sense that it provides the highest possible bandwidth for the given set of paths.

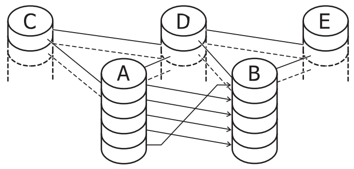


Fig. 6. Nodes split in graph.

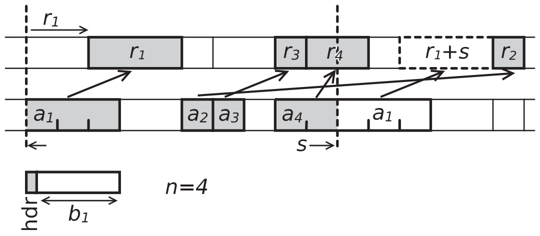


Fig. 7. Slot table wrap-around.

Algorithm 2. In-order path selection

```
Data : set of paths with given departure time slot, count of
  allocated slots and length
Result : reduced set of paths with in-order delivery
$s \leftarrow$ number of time slots;
Duplicate paths $p_1, \ldots, p_n$ as $p_{n+1}, \ldots, p_{2n}$ with delay $s$;
$solution \leftarrow \emptyset$;
for $\forall i \in \{1,2,\ldots,n\}$ do
  consider set $Q=\{p_i,\ldots,p_{i+n-1}\}$;
  $Q \leftarrow Q \backslash \{p_j \in Q | p_j$ arrives later than $p_{i+n-1}\}$;
  $Q$ is the working window;
  initialize $t_1,\ldots,t_{2n},t_{\emptyset} = 0$;
  for all flows $p_j \in Q$ do
    $best \leftarrow \emptyset$;
    for all flows $p_q \in Q, q < j$ do
      if $p_q$ arrives before $p_j$ and $t_q > t_{best}$
      $best \leftarrow i$;
      $predecessor_j \leftarrow q$;
    end
  end
  $t_j \leftarrow t_{best}$ + bandwidth of $p_j$;
  bf iIf $t_j$ is best solution so far then
      solution $\leftarrow$ solution reconstructed by following the
   chain of predecessors of $j$;
    end
  end
end
```

4.3.2. Proof of optimality for in-order path selection

In the following, by path we will refer to a path from source to destination and an associated set of contiguous available slots on that path.

Let A be the set of all paths $A = \{p_i | p_i \text{ is a path}\}$. In the following we will assume that A has at least one element.

Let us assume, without any loss of generality, that within a frame of the size of the slot table m, p_i departs before p_{i+1}, $\forall i \in \{1,2,\ldots,n-1\}$. We say without loss of generality because the indices i of p_i can be chosen in such a way that the departure times of paths $p_1,\ldots,p_n$ are chronologically ordered. Let $b_i > 0$ be the bandwidth delivered by path p_i and r_i the arrival time of path p_i when considering a particular slot table revolution starting at p_1 (Fig. 7). The bandwidth takes into account the header overhead.

Definition 1. A solution to the problem is a non-empty set $X \subseteq A$ which ensures in-order delivery.

We formalize the requirement for in-order delivery as:

$$\forall p_i, p_j \in X \text{ with } i < j \rightarrow r_i < r_j \wedge r_j < r_i + s$$

where s is the duration of one slot table revolution. Note that all sets containing a single path, that is, all sets of the form $X = \{p_q\}$ are solutions because a single path cannot produce out-of-order deliveries. Using the previous formalization we observe that the implication is always true since there is no $i < j$ among the valid indices.

Let ξ_X be the set of all solutions.

Let us denote $\mathscr{B}(X) = \sum_{\forall i}^{p_i \in X} b_i$

Definition 2. We call an optimal solution $\mathscr{A} \in \xi_X$, a solution that maximizes $\mathscr{B}(\mathscr{A})$

$$\mathscr{B}(\mathscr{A}) = \max_{X \in \xi_X} \mathscr{B}(X)$$

Let $\xi_{\mathscr{A}}$ be the set of all optimal solutions.

Let $X_i \in \xi_X$ be a solution with the property that $p_i \in X_i$, let ξ_{X_i} be the set of all solutions containing p_i, and $\mathscr{A}_i$ an optimum over the set ξ_{X_i}.

Lemma 1

$$\mathscr{B}(\mathscr{A}) = \max_{i \in \{1,\ldots,n\}} \mathscr{B}(\mathscr{A}_i)$$

Proof. We use the following property of the maximum:

$$\max_{x \in A \cup B} f(x) = \max(\max_{x \in A} f(x), \max_{y \in B} f(y))$$

We show that $\xi_X = \bigcup_{i \in \{1,\ldots,n\}} \xi_{X_i}$, Obviously $\xi_{X_i} \subseteq \xi_X, \forall i$ since ξ_X is the set of all solutions, and also $\bigcup_{i \in \{1,\ldots,n\}} \xi_{X_i} \subseteq \xi_X$. For the reverse implication, if $X \in \xi_X$, according to Definition 1, there is at least one element $p_j \in X$, but then $X \in \xi_{X_j}$ because ξ_{X_j} is the set of all solutions that contain p_j. □

This implies that by finding the maximum in each set ξ_{X_i} and selecting the highest value found, we obtain the global maximum. It also implies that the element in ξ_{X_i} for which this maximum is achieved is a global optimum.

Explanation: In Algorithm 2, the outer loop iterates over the local optima $\mathscr{A}_i$.

We show how the optimum can be found over the set ξ_{X_1}. The solution can be easily generalized since the table of slots is periodic and a period with the length of one slot table revolution can be chosen so that it starts with any of the paths p_j.

Explanation: In the practical implementation of the algorithm, this is achieved by duplicating the list of paths with the proper increment in arrival time and selecting n paths starting at position i. As an optimization, paths that are already known to conflict with path p_i in terms of order of arrival are already discarded at this point.

Let $Q_{1,j}$ be a subsets of A so that $Q_{1,j} = \{p_i \in A | 1 \leqslant i \leqslant j\}$.

Let $X_{1,j}$ with $j \in \{1,\ldots,n\}$ be a solution with the property that $X_{1,j} \subseteq Q_{1,j}$ and $p_1 \in X_{1,j}$ and $p_j \in X_{1,j}$. Let $\xi_{X_{1,j}}$ be the set of all such solutions.

Note that an $X_{1,j}$ does not necessarily always exist, as p_1 and p_j may produce out-of-order deliveries thus $\xi_{X_{1,j}}$ may be the empty set, but a solution exists at least for $j = 1$ which is $X_{1,1} = \{p_1\}$.

We define $\mathscr{A}_{1,j}$ as an optimal solution within the subset $\xi_{X_{1,j}}$, thus $\mathscr{B}(\mathscr{A}_{1,j}) = \max_{Y \in \xi_{X_{1,j}}} \mathscr{B}(Y)$

Since $\xi_{X_{1,1}}$ has only one element, which is $\{p_1\}$, $\mathscr{A}_{1,1} = \{p_1\}$.

Lemma 2. *For any $k > 1$, if $\mathscr{A}_{1,k}$ exists, we can compute $\mathscr{A}_{1,k} = \mathscr{A}_{1,j} \cup \{p_k\}$ by selecting $j < k$ which maximizes $\mathscr{B}_{\mathscr{A}1,j}$ with the restriction that p_j and p_k produce in-order delivery.*

Proof. We first prove that the in-order delivery condition for p_j and p_k is sufficient for ensuring in-order delivery for the entire set $\mathscr{A}_{1,k}$. If $\mathscr{A}_{1,j}$ is a solution, $r_j > r_i$, $\forall i < j \Rightarrow r_k > r_j > r_i$, $\forall i$ with the property that $p_i \in \mathscr{A}_{1,j}$. Since we are only interested in the case where $\mathscr{A}_{1,k}$ exists, $r_k < r_1 + s$, but $r_1 < r_i$, $\forall i$ with the property that $p_i \in \mathscr{A}_{1,j} \Rightarrow r_k < r_i + s$, for the same values of i, which is the second property required by Definition 1.

We prove by mathematical induction that this method of constructing $\mathscr{A}_{1,k}$ ensures the optimality criterion. $\mathscr{A}_{1,1} = \{p_1\}$ is obviously optimal, since when a single path is available no more bandwidth can be delivered than that provided by the path itself.

We prove the induction step by contradiction. Assume $\mathscr{A}_{1,1}, \ldots, \mathscr{A}_{1,k-1}$ are optimal sub-problem solutions as earlier described. If $\mathscr{A}_{1,k}$ were not an optimal solution, there exists $\mathscr{A}'_{1,k}$ so that $\mathscr{B}(\mathscr{A}'_{1,k}) > \mathscr{B}(\mathscr{A}_{1,k})$.

$\mathscr{A}'_{1,k}$ contains at least one element p_j where $j < k$. Let p_j be the element with the highest index $j < k$. Let $\mathscr{A}'_{1,j} = \mathscr{A}'_{1,k} \setminus \{p_k\}$. $\mathscr{A}'_{1,j}$ is a set containing only elements from $Q_{1,j}$ and provides in-order delivery because its superset $\mathscr{A}_{1,k}$ provides in-order delivery.

$\{p_j, p_k\}$ provides in-order delivery for the same reason, which implies that $\mathscr{A}_{1,j} \cup \{p_k\}$ respects the requirements for in-order delivery. It follows that $\mathscr{A}_{1,k}$ is at least as good a solution as $\mathscr{A}_{1,j} \cup \{p_k\}$ and as a result $\mathscr{B}(\mathscr{A}_{1,k}) \geqslant \mathscr{B}(\mathscr{A}_{1,j}) + b_k$, but $\mathscr{B}(\mathscr{A}'_{1,k}) > \mathscr{B}(\mathscr{A}_{1,k}) \Rightarrow \mathscr{B}(\mathscr{A}'_{1,j}) + b_k > \mathscr{B}(\mathscr{A}_{1,k}) \geqslant \mathscr{B}(\mathscr{A}_{1,k}) + b_k \Rightarrow \mathscr{B}(\mathscr{A}'_{1,j}) > \mathscr{B}(\mathscr{A}_{1,j})$ which is impossible, because $\mathscr{A}_{1,j}$ was already assumed to be an optimal solution to the $\xi_{X_{1,j}}$ subproblem (from a previous induction step, as $j < k$). □

Lemma 3

$$\mathscr{B}(\mathscr{A}_1) = \max_{i \in \{1,\ldots,n\}} (\mathscr{B}(\mathscr{A}_{1,i}))$$

Proof. We again use the property that: $\max_{x \in A \cup B} f(x) = \max(\max_{x \in A} f(x), \max_{y \in B} f(y))$. We show that $\xi_{X_1} = \bigcup_{i=1,\ldots,n} \xi_{X_{1,i}}$. It is first of all obvious that $\xi_{X_{1,i}} \subseteq \xi_{X_1}$ as any element of $\xi_{X_{1,i}}$ is a solution and it contains p_1, therefore $\bigcup_{i=1,\ldots,n} \xi_{X_{1,i}} \subseteq \xi_{X_1}$. For the proving the reverse implication $\xi_{X_1} \subseteq \bigcup_{i=1,\ldots,n} \xi_{X_{1,i}}$, consider an element $X_1 \in \xi_{X_1}$. Since n is a finite value, we can find $j <= n$ so that j is the highest index of an element $p_j \in X_1$, that is $\nexists k > j$, $p_k \in X_1$. It results that $X_1 \subseteq Q_{1,j}$, but at the same time we also know that $p_1 \in X_1$, $p_j \in X_1$ and X_1 is a solution, therefore $X_1 \in \xi_{X_{1,j}}$. □

This implies that by finding the maximum in each set $\xi_{X_{1,j}}$ and selecting the highest value found, we obtain the maximum over ξ_{X_1}. It also implies that the element in $\xi_{X_{1,j}}$ for which this maximum is achieved is a optimum over ξ_{X_1} and based on Lemma 1 also a global optimum.

4.3.3. Algorithm complexity

In this study we have used the exhaustive search as a basis for comparison for our multipath algorithm. This can represent a problem, since the number of possible paths between two nodes may be very large, even when the length of a path is bounded. For example, on a mesh network, the number of paths between two nodes situated at distance of m and n in the x and y directions respectively is $\binom{m}{m+n}$ which lays in the range of thousands for an 8×8 mesh, if the worst case corner-to-corner communication is considered, and can increase above one hundred million on a 16×16 mesh. Allowing paths longer than the minimal one also increases the number of choices.

It is also possible to use a heuristic with polynomial time complexity for the single-path allocation, as we have already demonstrated in [32]. In this case, the advantage of the flow algorithm is only higher when compared to the weaker heuristic algorithm. By replacing the branch-and-bound exhaustive search from the previous study with a backtracking technique we were able to easily run the algorithm for larger networks. If the size of the network is to be further increased beyond the possibilities of the exhaustive search we believe the heuristic to be a viable solution. The iterative multi-path search has the same complexity in asymptotic notation [35] as single-path algorithm, regardless of the algorithm chosen, assuming the number of paths is bounded by a constant.

In comparison, the advantage of the multi-path flow algorithm is that it runs in polynomial time complexity. Our implementation, based on optimizations by Edmonds and Karp [31] has a complexity of $O(ES^2)$ where E is the number of links and S the number of slots. The algorithm looks for a maximum of S augmenting paths. This is lower than the Edmonds–Karp bound of $O(VE)$ because of the particular structure of our graph, more precisely due to the fact that each augmenting path uses one more slot out of the total of S slots available. Each search for an augmenting path is performed in a graph with ES edges and has a complexity of $O(ES)$.

The in-order path selection algorithm consists of three nested loops performing a maximum of k iterations with k being the number of paths and therefore has a complexity of $O(k^3)$. Since the number of paths is bounded by the number of slots S, the worst-case complexity is $O(S^3)$. The in-order path selection algorithm has a negligible effect to the total run-time of the algorithm.

5. Experimental results

We have considered two sets of tests for our experiments. For both sets, the communication channels are between IPs randomly chosen with equal probability with the restriction that each IPs has to be connected by at least one communication channel. The bandwidth is chosen randomly with a uniform distribution in the range of 100–400 MBps. Mesh topologies of sizes varying between 4×4 and 8×8 were tested with a number of network interfaces between 1 and 4 connected to each router, without exceeding a number of 64 network interfaces in total.

The Æthereal network supports both multiple IPs per NI and multiple NIs per router. Using more NIs per router provides a higher bandwidth between the IPs and the actual network mesh. Although there is no intrinsic limit to the maximum number of NIs that one router can support, we choose a value of 4 because intuitively this allows a good floorplanning. It is also the value suggested by [36].

In our experiments the number of time slots was fixed to 32 and we used the default link width of 37 bits [37]. Twenty random usecases were constructed for each tested topology.

The first set consists of fixed-size usecases, in which 64 IPs are connected to networks of varying sizes and the required communication bandwidth is fixed regardless of the size of the network. This allows us to explore the tradeoff between area and performance, however it should be noted that an aliasing effect may be present, caused by the fact that the number of IPs cannot always be evenly divided among the available network interfaces.

In the second set the number of IPs is equal to the number of network interfaces and the average number of communication channels per IP (total count of ingoing plus outgoing) is three. This test allows us to compare how the relative performance of the algorithms scales on networks of varying sizes under similar load.

We use as a performance measure the working frequency of the network necessary to support the entire communication load. The frequency is determined using a binary search technique.

5.1. Fixed communication requirements

Fig. 8. With only two paths a large percentage of usecases produce a better result than with a single path available to each com-

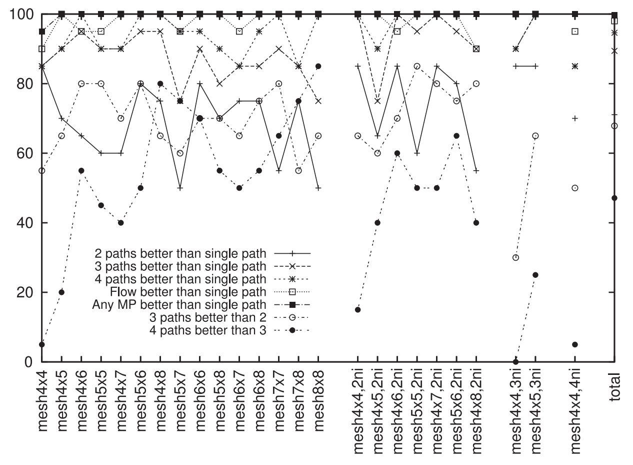


Fig. 8. Percentage of usecases in which one algorithm outperforms another.

munication channel. It should be noted even in this case not all channels are allocated using two paths, but only when the normal allocation fails. Further increases in the number of paths show diminishing returns.

The relative performance of the single path and multi-path allocation algorithms may show high variations depending on usecase. We plot a chart of the minimum frequency obtained by the single-path algorithm versus the minimum frequency obtained using the four paths and flow algorithm in Figs. 9 and 10 respectively. The highest reduction in frequency using four paths in a single usecase stands at 32.9% while the best result for flow algorithm was a reduction of 60.04%.

When analyzing the variation of performance across topologies of different sizes (Fig. 11), we find that when the communication requirements are fixed (and relatively high for the given topologies) the average improvement does not present high variations. One factor that clearly influences the relative performance of the single path and multipath algorithms is the number of NIs connected to the same router. This is because in networks with a single network interface per router it is more likely for the link from the network interface to the network to become congested first. Since this link has to be shared by all paths even in multi-path mode, that effectively sets a limit on the improvement that can be obtained.

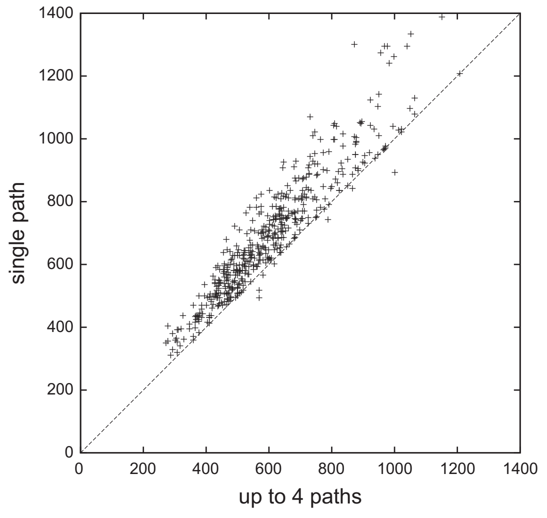


Fig. 9. Minimum frequency (MHz) allowed by the single-path allocation compared to allocation on four paths; each point represents one usecase-topology pair.

In order to give each usecase the same weight we have averaged after computing the ratio of frequencies. The ratio is computed as $(f_{high} - f_{low})/f_{high}$.

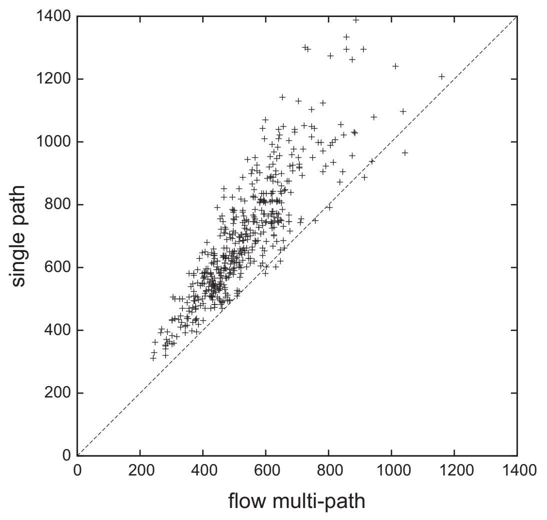


Fig. 10. Minimum frequency (MHz) allowed by the single-path allocation compared to the flow allocation; each point represents one usecase-topology pair.

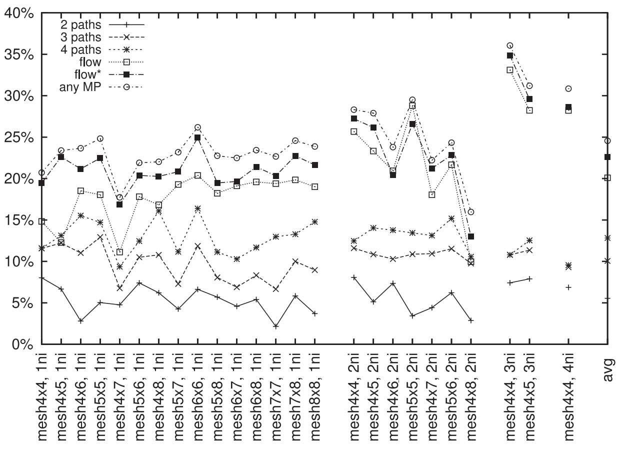


Fig. 11. Average reduction in frequency.

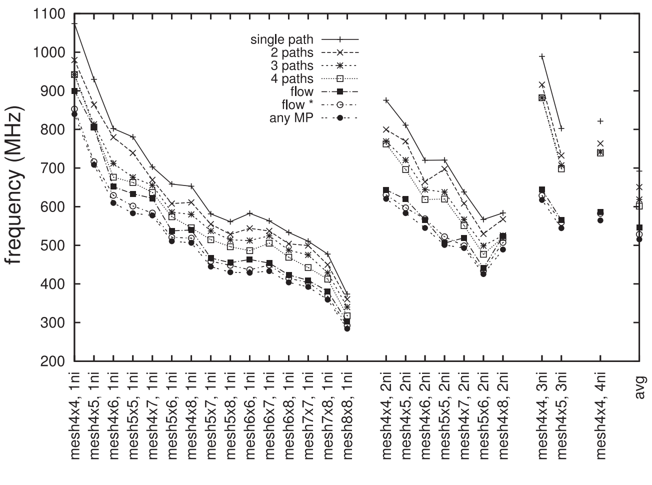


Fig. 12. Minimum running frequency that allows successful allocation of a usecase.

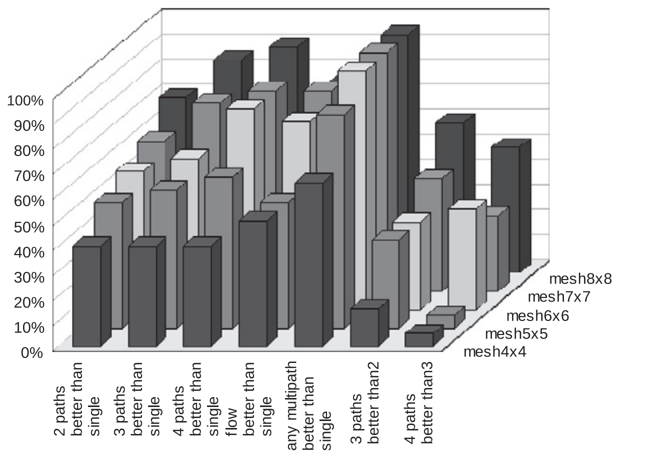


Fig. 13. Percentage of usecases on topologies of varying sizes in which one algorithm outperforms another.

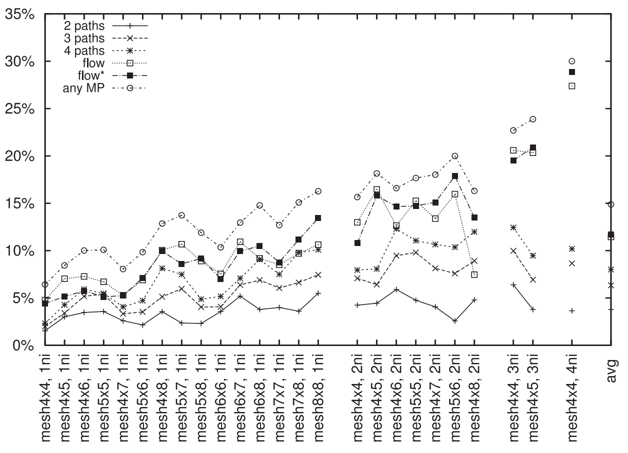


Fig. 15. Average reduction in frequency for each tested topology.

Fig. 12 presents the average frequency that is necessary for allocating one of the fixed-size usecases on a network of a given size. The figure shows how frequency can be traded for area. The same figure suggests that using the multipath approach may allow mapping usecases on smaller networks when a fixed frequency is considered. For example, using the multipath approach an usecase that would normally require a network of size 5×7 or 6×6 in order to run at 600 MHz can be mapped on a 5×5 network when using the multipath approach.

5.2. Scaled communication requirements

When considering communication requirements that are scaled with the size of the network, we find that for similarly loaded networks the relative performance of the multipath algorithm increases with network size. This can be explained by the fact that larger networks provide a larger path diversity.

The percentage of usecases which produce a better result in terms of frequency using the multipath approach is presented in Figs. 13 and 14. For small networks especially, the benefit of a larger number of paths is reduced, in particular, for the 4×4 and 5×5 meshes using a maximum of four paths instead of three only produced a benefit in 1 out of 20 usecases. The trend indicates that for larger networks increasing the number of paths even further can still be beneficial.

In terms of frequency reduction, we can observe a similar trend in Fig. 15. The observation that larger gains are obtained for networks with more network interfaces connected to the same router also remains true in this case. Under the decreased load compared with the fixed usecase, the highest improvement obtained with the four-paths algorithm was of 31.42% and with the flow algorithm of 51.66%.

5.3. Algorithm running time

One additional concern is the running time of the algorithm, as the complexity of exhaustive search is exponential with the distance between nodes. In our tests this was not found to be a problem as the average running time stayed below 1 s, even for the larger topologies (Fig. 16). The larger spike for the $7 \times 7, 7 \times 8$ meshes was caused by single usecases which ran for 28 and 12 s respectively.

6. Conclusions and future work

In this study we have analyzed the performance of a slot allocation tool based on multipath routing, in conjunction with the hardware costs and benefits from employing such a technique. We have found our technique to allow reductions in terms of frequency of up to 60.04%, with an average of 24.55% over all considered usecases. The benefit can also be expressed in terms of area savings

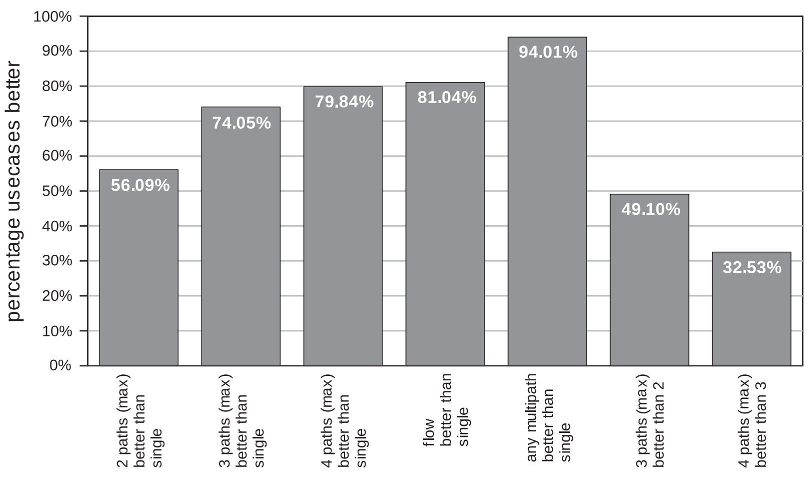


Fig. 14. Percentage of usecases over all scaled tests in which one algorithm outperforms another.

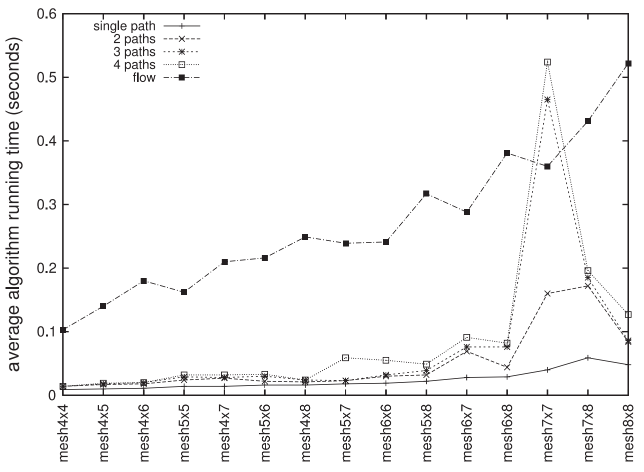


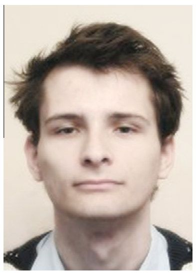
Fig. 16. Average algorithm running time.

by allowing the same communication requirements to be satisfied by a network of smaller size.

Although multipath routing may present significant challenges in the case of a generic packet switched NoC we show that it can be implemented with moderate cost in the Æthereal circuit switched TDM NoC.

References

[1] W.J. Dally, B. Towles, Route packets, not wires: on-chip interconnection networks, in: DAC, 2001.
[2] G. De Micheli, L. Benini (Eds.), Networks on Chips: Technology and Tools, The Morgan Kaufmann Series in Systems on Silicon, Morgan Kaufmann, 2006.
[3] K. Goossens, J. Dielissen, A. Radulescu, Æthereal network on chip: concepts, architectures, and implementations, IEEE Design & Test of Computers 22 (5) (2005).
[4] T. Bjerregaard, J. Sparso, A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip, in: DATE, 2005.
[5] M. Millberg, E. Nilsson, R. Thid, A. Jantsch, Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip, in: DATE, 2004.
[6] K. Goossens, A. Hansson, The aethereal network on chip after ten years: goals, evolution, lessons, and future, in: DAC, 2010.
[7] S. Vutukury, A simple approximation to minimum-delay routing, in: ACM SIGCOMM, 1999.
[8] I. Cidon, R. Rom, Y. Shavitt, Analysis of multi-path routing, IEEE/ACM Transactions on Networking 7 (6) (1999).
[9] I. Cidon, R. Rom, Y. Shavitt, Multi-path routing combined with resource reservation, INFOCOM 1 (1997).
[10] S. Murali, D. Atienza, L. Benini, G. De Micheli, A method for routing packets across multiple paths in NoCs with in-order delivery and fault-tolerance guarantees, VLSI-Design Journal (2007).
[11] L.M. Ni, P.K. McKinley, A survey of wormhole routing techniques in direct networks, Computer 26 (2) (1993).
[12] J. Hu, R. Marculescu, DyAD: smart routing for networks-on-chip, in: DAC, 2004.
[13] M. Li, Q.-A. Zeng, W.-B. Jone, DyXY: A proximity congestion-aware deadlock-free dynamic routing method for network on chip, in: DAC, 2006.
[14] G. Ascia, V. Catania, M. Palesi, D. Patti, A new selection policy for adaptive routing in network on chip, in: EHAC, 2006.
[15] I.-G. Lee, J. Lee, S.-C. Park, Adaptive routing scheme for NoC communication architecture, in: ICACT, 2005.
[16] M.A.A. Faruque, T. Ebi, J. Henkel, Run-time adaptive on-chip communication scheme, in: ICCAD, 2007.
[17] Z. Lu, A. Jantsch, TDM virtual-circuit configuration for network-on-chip, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 16 (8) (2008).
[18] Z. Lu, A. Jantsch, Slot allocation using logical networks for TDM virtual-circuit configuration for network-on-chip, in: ICCAD, 2007.
[19] J. Liang, S. Swaminathan, R. Tessier, aSOC: a scalable, single-chip communications architecture, in: PACT, 2000.
[20] J.M. Nolen, R.N. Mahapatra, Time-division-multiplexed test delivery for NoC systems, IEEE Design & Test 25 (1) (2008).
[21] Y. Wang, K. Zhou, Z. Lu, H. Yang, Dynamic TDM virtual circuit implementation for NoC, in: APCCAS, 2008.
[22] P. Wolkotte, G. Smit, G. Rauwerda, L. Smit, An energy-efficient reconfigurable circuit-switched network-on-chip, in: IPDPS, 2005.
[23] A. Hansson, K. Goossens, A. Rădulescu, A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic, VLSI Design (2007).
[24] S. Murali, G.D. Micheli, SUNMAP: a tool for automatic topology selection and generation for NoCs, in: DAC, ACM, 2004.
[25] J. Liu, L.-R. Zheng, H. Tenhunen, Interconnect intellectual property for network-on-chip (NoC), Journal of Systems Architecture 50 (2-3) (2004).
[26] A. Hansson, M. Coenen, K. Goossens, Channel trees: reducing latency by sharing time slots in time-multiplexed networks on chip, in: CODES+ISSS, 2007.
[27] B. Gebremichael, F. Vaandrager, M. Zhang, K. Goossens, E. Rijpkema, A. Rădulescu, Deadlock prevention in the Æthereal protocol, in: CHARME, 2005.
[28] P.M. Merlin, J. Schweitzer, Deadlock avoidance in store-and-forward networks-11: other deadlock types, IEEE Transactions on Communications 28 (3) (1980).
[29] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, H. Corporaal, Dynamic time-slot allocation for QoS enabled networks on chip, in: ESTIMedia, 2005.
[30] O. Moreira, J.J. Mol, M. Bekooij, Online resource management in a multiprocessor with a network-on-chip, in: ACM Symposium on Applied Computing, 2007.
[31] J. Edmonds, R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, Journal of the ACM 19 (2) (1972).
[32] R. Stefan, K. Goossens, Multipath routing in TDM NoCs, in: VLSI-SoC, 2009.
[33] C. Schensted, Longest increasing and decreasing subsequences, Canadian Journal of Mathematics 13 (1961).
[34] M.H. Albert, M.D. Atkinson, D. Nussbaum, J.-R. Sack, N. Santoro, On the longest increasing subsequence of a circular list, Information Processing Letters 101 (2) (2007).
[35] D.E. Knuth, Big omicron and big omega and big theta, SIGACT News 8 (2) (1976).
[36] J. Balfour, W.J. Dally, Design tradeoffs for tiled CMP on-chip networks, in: ICS, 2006.
[37] A. Hansson, A Composable and Predictable on-chip Interconnect, Ph.D. Thesis, Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands, 2009.



Radu Stefan has received his BE degree from Transilvania University of Brasov in 2005 with FPGA configuration compression as his thesis subject and his MSc degree from the same university in 2006 with a final dissertation on training neural networks using genetic algorithms. In 2005–2006 he worked for Splash Software Romania on rate-control in high-end video compression and MPEG4 encoder parallelization. He is currently a PhD candidate at the Delft University of Technology where his research focuses on resource allocation in Networks-on-Chip.

Kees Goossens received his PhD from the University of Edinburgh in 1993 on hardware verification using embeddings of formal semantics of hardware description languages in proof systems. He worked for Philips/NXP Research from 1995 to 2010 on networks on chip for consumer electronics, where real-time performance and low cost are major constraints. He was part-time full professor at the Delft university of technology from 2007 to 2010, and is currently full professor at the Eindhoven university of technology, where his research focusses on composable (virtualised), predictable (real-time), low-power embedded systems.