Wydział Informatyki
Zachodniopomorski Uniwersytet Technologiczny

Praca magisterska

Model transmisji strumieni o gwarantowanej oraz możliwie najwyższej przepustowości w sieciach wewnątrzukładowych

Andrzej Godziuk
`andrzej@godziuk.pl`

Promotor: Dr inż Piotr Dziurzański

Szczecin, 2009

Faculty of Computer Science & Information Systems
West Pomeranian University of Technology

Master Thesis

Guaranteed Throughput and Best Effort Streams in a Network on Chip Model

Andrzej Godziuk
`andrzej@godziuk.pl`

Supervisor: Dr inż Piotr Dziurzański

Szczecin, 2009

Abstract

Wraz z rosnącą złożonością układów scalonych z dużą liczbą rdzeni, rodzi się potrzeba skalowalnej architektury. Jedno z rozwiązań to sieć wewnątrzukładowa opierająca się na przełączaniu pakietów.

Zaproponowano algorytm routowania dla sieci o topologii dwuwymiarowej siatki, która dba o balansowanie obciążenia zachowując również rygory jakości transmisji używając kanałów wirtualnych dla obsługi zróżnicowanych priorytetów pakietów.

With the growing complexity of many-core chips, a scalable architecture is required. One of the solutions is packed-switched network-on-chip (NoC).

A routing algorithm for 2D-mesh NoC is proposed, which performs load balancing while maintaining quality-of-service (QoS) using virtual channels for packet priorities.

Contents

Chapter 1

Introduction

1.1 Background

According to the Moore's Law [1], every two years the number of transistors that can be fit on a single integrated circuit (IC) chip doubles. This allowed creation of more and more complex ICs, which has led to the latest trend called System on Chip (SoC), where many cores - which previously have been distributed on many chips - are placed on a single silicon chip and enclosed in a common case.

As more and more cores were placed on a chip, the interconnections of increasing complexity had to be developed. First such systems included direct connections between each of the cores[2]. When the number of cores increased, this solution became inefficient in terms of both occupied space and energy[2].

The approach that solved these problems was a shared system bus that was connected to each of the cores[3]. Although this allowed many more cores to be fit into a single package, their number were limited by bus bandwidth that had to be large enough to allow each of the cores access to push its data over it.

Several solutions are proposed to overcome the limitations of on-chip bus, one of which is Network on Chip (NoC). It can be thought of as an attempt to adapt packet-switched networks to NoC conditions.

1.2 General concepts

Although many architectures of NoC have been proposed, there is a common part.

Every core placed on chip is accompanied by a network interface (NIC) connected directly to a router using a two-way bus[4]. A core with accompanying NIC and router is called a network node.

Every router is connected to some or all other routers in a manner that is called a network topology.

Routers have input ports and output ports. Their purpose is to direct packets from input ports to output ports using a predefined routing algorithm. There are many classes of routing algorithms implemented in routers, but two main can be distinguished: static (oblivious) and dynamic algorithms[5]. The former direct packets from node A to node B using always the same path, while the latter can direct packets using various paths in a random or adaptive manner. Packets are divided into flits, which are an unit of transfer between routers[6].

In a sufficiently complex system, several classes of packets may be needed. The most common example is to distinguish between high-priority low-latency packets used i.e. for signaling and bulk transfer where latency doesn't matter that much. Networks that handle such classes of packets are said to guarantee Quality of Service (QoS)[4], and in context of NoC, such networks are often called Quality NoC, or QNoC.

Moreover, NoC allow designers to achieve a higher level of flexibility than other approaches and, therefore, higher resource reuse. An example of what would be impossible or overly complex with traditional architectures is live reconfiguration of SoC, with undisrupted QoS[7].

1.2.1 2-mesh networks

2-mesh is a topology where each of the network nodes is connected to 4 other nodes in 4 directions: north, east, west and south. Each of these links is bidirectional. This keeps the channel length short but can cause load imbalance for many traffic patterns, as a mesh network has half the bisection bandwidth of a torus with the same radix and dimension [8].

2-mesh networks are the most commonly used NoC topology [9]. This is because of the way they map onto physical chip structure - which is also two-dimensional and the nodes are roughly rectangles. Because of this, they are superior in terms of scalability and ease of synthesis. Even fully automatic design flows have been proposed [10][11].

A packet can be sent from node A to node B via many different routes. Each route has a number of hops, which is count of routers that have forwarded the packet during its way from source to destination. A subset of routes, called minimal routes, have the least number of hops. In a network with little bandwidth used by actual traffic, minimal routes would be the quickest. In real networks, however, due to large occupancy of some (or all) segments of a route by other traffic, non-minimal routes can be faster than minimal ones.

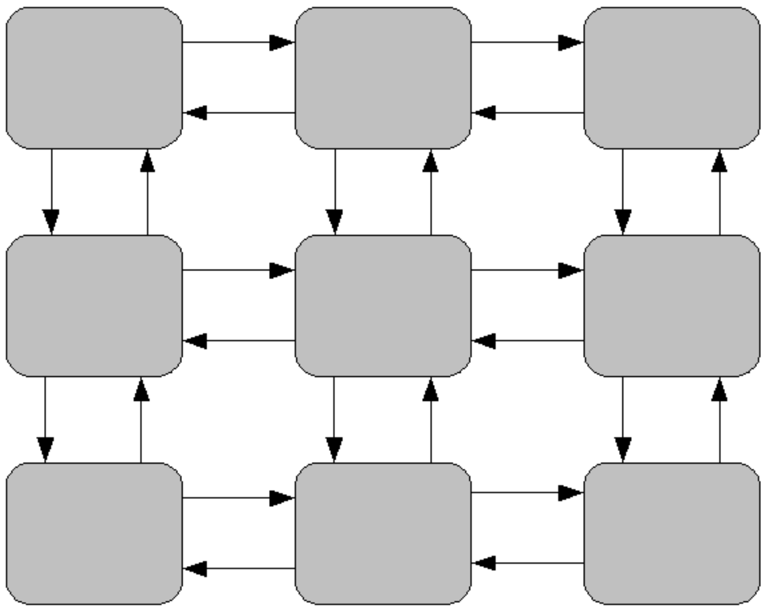


Figure 1.1: An example 3x3 2-mesh Network on Chip

Regions

Because core sizes often differ, in 2-meshes size of all cells needs to be the size of largest cell, which is inefficient in terms of both space and consumed energy [12]. Therefore, a concept of regions - cores taking the size of several cells with no routers inside - has been introduced. Despite their similarity to 2-meshes, mesh networks with regions need different routing algorithms [13].

1.2.2 Routing in 2-mesh networks

The simplest approach to routing in mesh networks is Dimension-Order Routing. For 2-mesh, this method is called XY-routing (or YX-routing for another variant of the same method).

When a packet arrives to the router, it first checks if its own X-coordinate is the same as X-coordinate of the destination node. If not, the router forwards the packet to appropriate output port - east or west. If the X-coordinate is the same as destination, it forwards the packet to north/south output port[8].

The XY routing algorithm is therefore a deterministic algorithm, as the packet from node A to node B always uses the same route. Its greatest advantage the routers are very small while retaining minimal routes. The disadvantage is

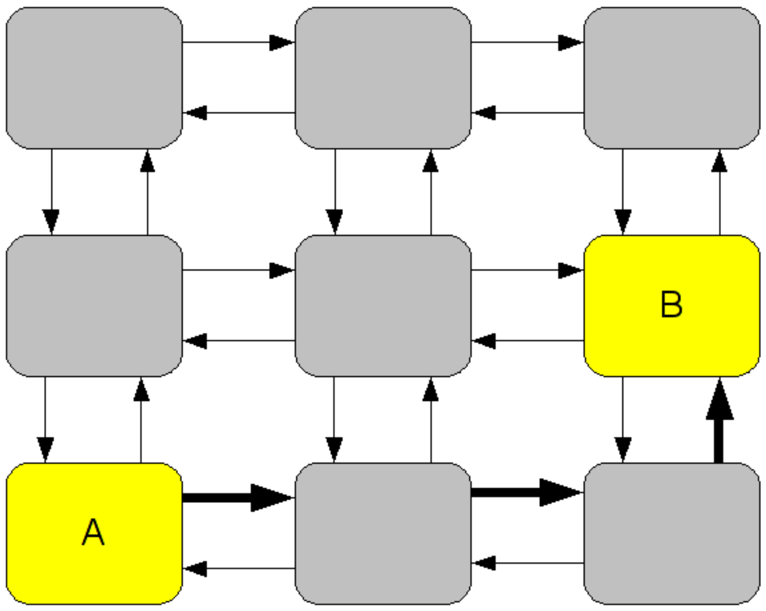


Figure 1.2: Route of a packet from node A to node B using XY routing

poor load balancing.

One variation of XY algorithm is Valiant's algorithm, which first routes a packet to a randomly-chosen node, and then routes it to its destination node. This improves load balancing by reducing the load of any traffic pattern to half of the capacity of a network. The obvious disadvantage of Valiant's algorithm is that it uses non-minimal routes, so latency can be high and additional bandwidth is used for routing to the intermediate node[8].

Simple variations of XY algorithm have been developed to minimize the risk of hotspots in certain traffic patterns. Toggle XY (TXY), in which packets are split evenly between XY and YX routing, is one example. A weighted variation of TXY exists, where each node has a XY/YX probability ratio is loaded into a router at programming time. Source Toggle XY is an algorithm where, at each node, XY or YX is chosen based on bitwise XOR function of source and destination ID. In Weighted Ordering Table algorithm, each source stores a routing bit that determines XY/YX for each destination. The bits are set at programming time, which allows manual optimization[14].

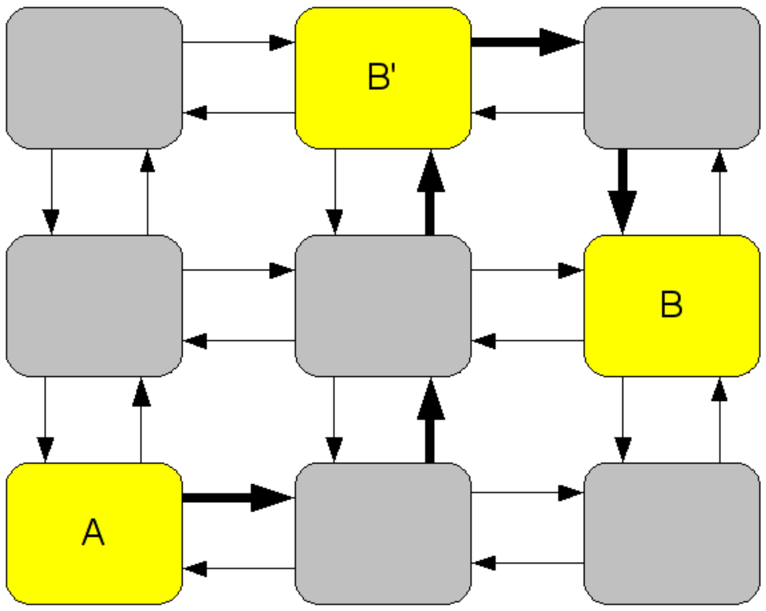


Figure 1.3: Routing of a packet from node A to node B using randomly-chosen intermediate node B’

Hot spots

Often when certain traffic patterns occur, packets from different connections may be routed via the same routers, while leaving the other ones idle. This leads to inefficient usage of network bandwidth and high contention level [15] and can be avoided using different algorithms, Valiant being the simplest example. Apart from inefficiently using bandwidth, hot spots may cause problems with heat distribution, focusing most heat in one place.

1.2.3 QoS in on-chip networks

Quality of Service (QoS) is often introduced in NoC. The concept is to divide the traffic into several different priority classes that have guarantees regarding maximum latency and/or bandwidth.

The concept of QoS has been introduced in computer networks and is a well-known concept. It is widely used in latency-sensitive applications like Voice over IP, online games and Internet Protocol Television. Sophisticated algorithms have been designed and implemented in software to achieve these goals for computer networks QoS. However the basics of QoS in on-chip networks are similar, the reason for using these mechanisms is different, and the algorithms must be simple enough to be resource-effective, as they are implemented in hardware.

The main reason of introducing QoS in NoC is to make the chip perform its task using less network bandwidth. In a simple NoC, the network must be fast enough to ensure that the most latency-demanding packets will be sent within the time limit even in worst case. This could lead to a situation where network resources are idle most of the time, just to ensure efficiency even in worst case. Because the most latency-sensitive packets are small and used for signalling only, increasing their priority will not affect network performance severely, and much less bandwidth will be needed to ensure proper chip operation.

There are basically two physical values impacting network bandwidth: clock speed and channel width. While clock speed affects chip's internal structure, it is usually constant and it's the channel width that takes most physical resources. Reducing network bandwidth therefore results in widening channel width, which results in less space taken by interconnection infrastructure and less heat produced. On the other hand, we reduce resources taken by network channels by introducing overhead of QoS logic in routers. This is the reason why new, resource-efficent QoS algorithms must be developed for NoC - otherwise introducing QoS would unnecessarily complicate the chip structure.

While QoS is mainly achieved by proper construction of queues and proper allocation of resources to virtual channels, it can become more efficient when backed by a routing algorithm aware of QoS demands [16].

TDMA

Time division multiple access is among the simplest QoS methods. It involves dividing channel into several time-slots, each core assigned to one or more slots. It has hard real-time guarantees, however, it allows situations where channel is unused when there is demand for it, so it doesn't use bandwidth efficiently losing all the advantages of QNoC.

When using TDMA, one must chose betwen either over allocation, where each core gets enough bandwidth to satisfy its peak needs, or under-allocation, where the amount of bandwidth each core gets, is under its production capacity and therefore latency increases under heavy load.

1.3 Related work

1.3.1 Æthereal

Æthereal is a network developed at Philips Research Laboratories to achieve several goals such as uncorrupted, lossless, ordered data delivery; guaranteed throughput; and bounded latency[17]. Æthereal consists of hardware, a programming model and a design flow. Research is being made to allow Æthereal span across multiple chips[18]. It is currently used in Philips television receivers.

Æthereal uses reconfigurable Time-Division Multiplexing at a router level. Time is divided into S slots and routers keep a table telling them, for every slot number, packets from which inputs should be directed to which outputs. This creates time-divided circuit switching - at a given time slot, it is guaranteed that a packet from input i_m will be redirected to output o_n. If all the routers are pre-programmed with a route, when a packet is injected at a right time slot, it is guaranteed to be redirected to a destination node via a known route, with a known delay.

Best-effort packets carry their route in headers and are forwarded in time slots not taken by Guaranteed Throughput (GT).

Before a GT route is used, routers must fill their tables to contain entries allowing for the packet to be sent. This may be done in programming time or at runtime. When set up at runtime, three types of packets are used: SetUp, AckSetUp and TearDown. They are implemented as BE packets, containing the route to be set up. When a SetUp packet arrives at a router input i_m at time slot s, the router marks that every time at slot s it will forward the packet from input i_m to the desired output, actually forwards the packet, and the next router sets it table at slot $s+1$. When connection creation succeeds, a AckSetUp packet is sent to the source; otherwise a TearDown packet is sent, undoing any changes in routing tables.

In the example on Figure 1.4, routers and their routing tables are shown in the slot s=2. Let's consider route b. At time slot s=2, router R_1 will forward packet from input i_1 to output o_2. At slot 3, this packet will be routed at R_2 from i_0 to o_3. In a similar manner one can analyze route a. At time slot s=1, a packet arrives at i_0 of R_1, and is forwarded to o_2. At time slot s=2, it is forwarded from i_0 to o_2 at router R_1. At time slot s=3, it is forwarded from i_0 to o_2 at router R_2.

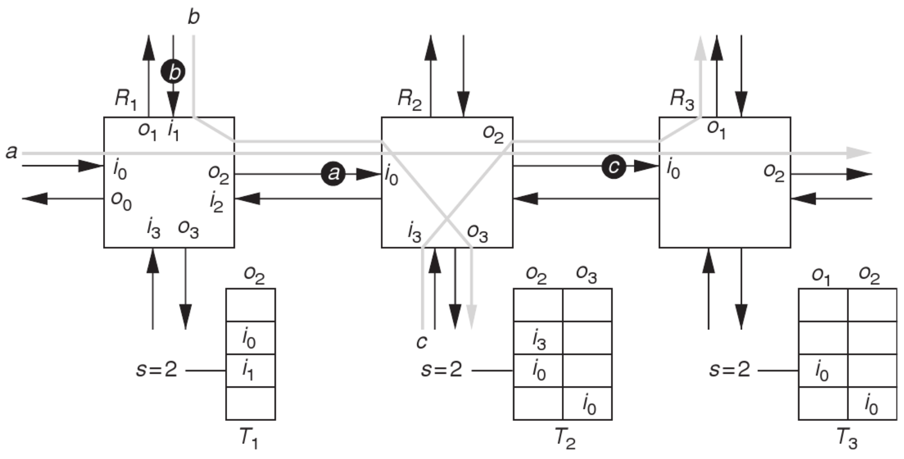


Figure 1.4: Example of routing in Æthereal network [17]

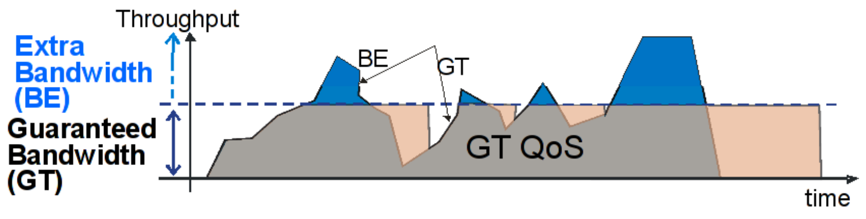


Figure 1.5: Using more bandwidth than guaranteed increases latency[19]

1.3.2 SuperGT network

SuperGT network has been introduced to satisfy demands of scalable video coding. It provides minimum guaranteed traffic as well as non-guaranteed extra traffic [19].

The network uses TDMA for guaranteed-throughput as well as division of packets into two priorities called Lo and Hi. Time is divided into slots, some of which are assigned for high-priority traffic only. Time division is handled at network interface level. Networks interfaces may have three different QoS levels assigned: Guaranteed Throughput for interfaces that are allowed to inject only during time slots allocated for high priority packets, Best Effort for interfaces that are allowed to inject packets only during low-priority packet time slots and SuperGT for interfaces that are allowed to inject packets during both types of time slots.

Routers in SuperGT a network have two virtual channels, one called superGT, which deals with both Hi and Lo priority packets, and another one called escape-High, which handles only Hi priority packets. Virtual channels are accompanied by management logic which allows switching priority of packets in superGT channel in order to ensure in-order packet delivery.

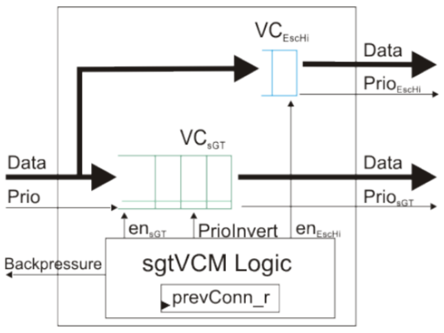


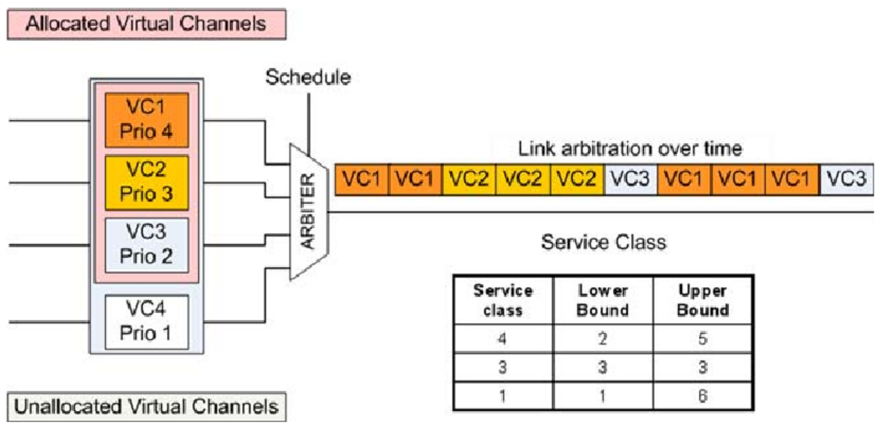
Figure 1.6: SuperGT routers employ two virtual channels and Channel Manager logic [19].

1.3.3 Network with BAA arbitration

The Bounded Arbitration Algorithm (BAA) has been developed to join the best of TDMA-based and packet switching-based NoC. Packet-switched networks achieve high bandwidth utilization while failing to provide 100% guarantees, while TDMA-based networks provide hard guarantees at a cost of low bandwidth utilization. BAA-based networks provide hard parameter guarantees while keeping bandwidth utilization averagely as high as 97% [20].

Source-based deterministic routing is used utilizing the XY algorithm. Network architecture supports a large number of fine-grained packet priorities. Two bounduary priorities exist: Prio H, guarantying low latency with no bandwidth or jitter guarantee, and Prio L, providing no guarantees regarding latency, bandwidth or jitter. Between the two, a large number of well-defined priority classes exist. Each class defines lower and upper bounduaries for bandwidth and latency, jitter problem is handled by keeping both bounds the same. The bounduaries are expressed in packets/cycle.

The Bounded Arbitration Algorithm has been developed. First, the algorithm satisfies lower bounds for each of the virtual channels, ordering by priority, then it continues to route traffic remaining to the upper bounds.


Allocated Virtual Channels
Schedule
VC1 Prio 4
VC2 Prio 3
VC3 Prio 2
VC4 Prio 1
ARBITER
Link arbitration over time
VC1 VC1 VC2 VC2 VC2 VC3 VC1 VC1 VC1 VC3
Service Class
Unallocated Virtual Channels

Service class	Lower Bound	Upper Bound
4	2	5
3	3	3
1	1	6

Figure 1.7: Bounded Arbitration Algorithm: The arbiter first satisfies lower bounds, then proceeds to satisfy upper bounds [20]

1.4 Conclusion

Currently existing QNoCs provide both Best-Effort and Guaranteed Service streams, they however use XY routing or one of its variations, not caring about load balancing. Currently existing load-balancing algorithms for 2D meshes are not QoS-aware and therefore cannot be used in networks providing hard guarantees.

A QoS-aware load-balanced routing algorithm is proposed in this thesis. The author believes that such algorithm would increase bandwidth utilization and reduce the risk of hot spots.

Chapter 2

Proposition: Priority-aware routing

Hereby, a new routing algorithm is proposed. Its purpose is to avoid creation of hotspots in networks with packet priorities. It treats packets with different priorities in different ways, allowing packets with lower priority to be routed on a non-minimal route, and the higher the priority is, the less extra hops are allowed.

2.1 Algorithm description

This algorithm needs one more signal between routers, as seen in figure 2.1. The signal informs the source router that the input queue of destination router is close to full. This enables the source router to take a decision to put some of the packets on a route that avoids sending data in that particular direction.

The algorithm also uses an extra field in a packet, being a 2-5 bit unsigned integer. The field is called N_{bad} and keeps the number of hops when packet can be directed on non-minimal routes. Its initial value is set by source router, and is a fixed number depending on packet priority. For highest priority it could have a value of 0, therefore enforcing that the packet traverses minimal routes only.

In more formal terms, each router R_m has an input signal QFULL from each of four routers in its neighbourhood. Router R_n in neighbourhood of router R_m sets signal QFULL to "true" when number of flits S_n kept in input queue of R_n is larger than some critical fraction R_{crit} of its maximum size S_{max}:

$$S_n > R_{crit} \cdot S_{\max} \qquad (2.1)$$

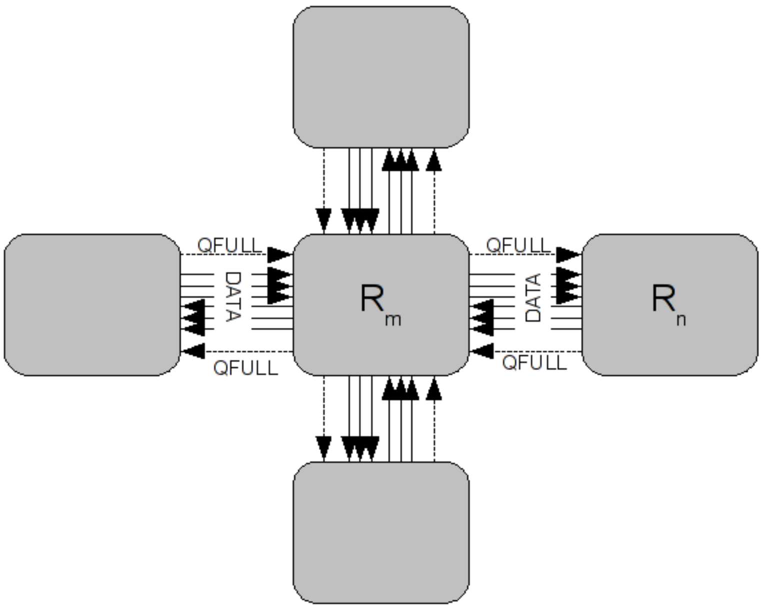


Figure 2.1: The algorithm requires one control signal - QFULL

When chosing the direction of next hop, the router first computes directions for two possible minimal routes, based on XY and YX algorithms (1, 3). If any of them does not have QFULL signal set, it becomes the final route (2, 4).

In the opposite case, N_{bad} field of a packet is checked (5). If it is zero, the packet gets routed to a direction based on XY algorithm (7), if not - its direction is one of the two directions not computed by XY/YX (6).

This has been formally described as Algorithm 1.

```
Input: N_bad - number of bad hops left
Input: QFULL_N, QFULL_S, QFULL_W, QFULL_E
Output: D - routing direction

D_X = result of XY(Flit);
D_Y = result of YX(Flit);
1 if not $QFULL_{D_x}$ then
2     D = D_X ;
  end
  else
3     if not $QFULL_{D_y}$ then
4         D = D_Y;
      end
      else
5         if $N_{bad} > 0$ then
              foreach direction in ('N', 'S', 'W', 'E') do
                  if direction not in ($D_x$, $D_y$) then
6                     D = direction;
                  end
              end
          end
          else
7             D = D_X;
          end
      end
  end
```

Algorithm 1: Priority-aware routing (PARouting)

2.1.1 Parameters of algorithm

The algorithm has two parameters which can be tuned. Effect of these parameters on routing efficency and load distribution is to be measured and presented further in this thesis.

"Queue-near-full" critical value

The ratio $R_{crit} = \frac{S_{crit}}{S_{max}}$ controls the level of queue occupancy at which signal QFULL gets emited. Its ideal value depends on traffic pattern and mesh size. The expected result is increase of load distribution with R_{crit} approaching 100%.

Maximum number of ”bad hops”

Each priority has assigned a number of bad hops (hops on non-minimal routes) that are allowed to the packet of that priority. The more these values will differ, the better expected result for high-priority packets will be.

2.2 Older version

Before the algorithm above has been developed, another version has been measured. In this thesis it is called PA2Routing. It is more complicated than PARouting and has worse results in most tests.

2.2.1 Algorithm description

This algorithm has one more input signal: direction from which the routed filt came (1). It has been formally described as Algorithm 2.

```
Input: $N_{bad}$ - number of bad hops left
Input: $D_{in}$ - direction of input port
Input: $QFULL_N$, $QFULL_S$, $QFULL_W$, $QFULL_E$
Output: D - routing direction

$D_X$ = result of XY(Flit);
$D_Y$ = result of YX(Flit);

if not $QFULL_{D_x}$ then
    $D = D_X$ ;
end
else
    if not $QFULL_{D_y}$ then
        $D = D_Y$;
    end
    else
        if $N_{bad} > 0$ and $QFULL_{D_x}$ and $QFULL_{D_y}$ or $D_x = D_{in}$ or
        $D_y = D_{in}$ then
            $D_{pool}$ = empty set
            foreach direction in ('N', 'S', 'W', 'E') do
                if direction = $D_{in}$ or there is no such output then
                    skip this direction
                end
                Append direction to $D_{pool}$
            end
            D = random element of $D_{pool}$
        end
        else
            D = random element of set ($D_X$, $D_Y$)
        end
    end
end
```

Algorithm 2: Priority-aware routing, earlier version (PA2Routing)

Chapter 3

Simulation environment

The network has been simulated in a custom network simulator written especially for this thesis.

Simulator provides a general framework performing routing, generating load and tracing packet flow. All actions done to a packet can be logged and registered for statistics. After the simulation has been completed (a number of simulation cycles has passed), it computes statistics basing on gathered data.

3.1 Routing

The framework provides an abstract router class, which should be extended by user's routing algorithm. The class itself provides code for gathering statistical data and actually routing packets basing on direction computed by routing algorithm. The algorithm is provided as a Python mix-in.

Routers have three input queues, for three packet priority classes. Each router also has four inputs and four outputs. On modelling level, each router keeps references to its neighbouring routers. Routers pass flits between each other using defined interfaces, therefore they are performing Transaction-level Modeling.

Routers gather the following statistical data: maximum input queue size encountered during simulation and number of packets sent broken down by direction.

3.2 Load generation

For each simulation, one instance of load generator is created. Its purpose is to inject packets to routers' input queues (therefore simulating input from IP core associated with the router). Routers where packets are injected are chosen using arbitrary random function.

The framework provides an abstract class for load generator, which is then extended by traffic pattern algorithm. Traffic patterns are

3.3 Scheduler

A simple scheduler has been used which routes packet from first non-empty queue, iterating queues of virtual channels starting from highest priority. It is the reason why in experiments, even XY routing gives noticable difference in latency between priorities.

Chapter 4

Simulation study

4.1 Impact of traffic patterns on routing algorithms performance

4.1.1 Experiment conditions

An experiment has been conducted on 4x4 mesh during 10000 simulation ticks to measure impact of traffic patterns on algorithm performance parameters.

In order to enforce hot spots in the network, for each generated packet, both coordinates of its source address are chosen using Gauusian random function with $\mu = 2.5$, $\sigma = 0.9$ with saturation function applied afterwards.

For each source address, a destination address is computed using one of three commonly used [8] traffic patterns: Reverse (4.1), Shuffle (4.2) and Tornado(4.3). In the definitions below, the following variable are used:

- w, h - width and height of the network
- x_s, y_s - source address
- x_d, y_d - destination address

$$x_d = w - x_s - 1, y_d = w - y_s - 1 \tag{4.1}$$

$$x_d = (2 \times x_s) \bmod w, y_d = (2 \times y_s) \bmod h \tag{4.2}$$

$$x_d = (x_s + w \div 2 - 1) \bmod w, y_d = (y_s + h \div 2 - 1) \bmod h \tag{4.3}$$

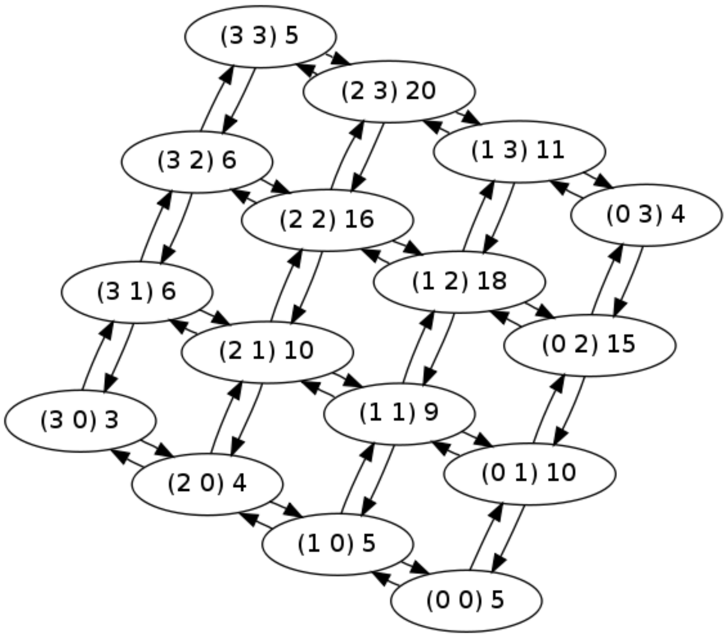


Figure 4.1: Reverse traffic pattern, XY router - maximum queue length 20

4.1.2 Impact on queue size

Minimum required queue size has been measured. This has been achieved by allowing algorithm to have arbitrary queue size and measuring its top size during simulation.

XY routing

Queue sizes for XY routers in 4x4 mesh with various traffic patterns have been measured: Reverse (Figure 4.1), Shuffle (Figure 4.2) and Tornado (Figure 4.3).

The numbers in figures are as following: $(x\ y)\ Q_{len}$, where x and y are router's coordinates and Q_{len} is maximum recorded queue size.

Shuffle traffic pattern caused the biggest queue sizes in the set of patterns under experiment. Standard deviation of queue sizes has been also calculated for different traffic patterns and was as following: Reverse - 5.27 , Shuffle - 10.32 , Tornado - 4.89 .

Priority-aware routing

Queue sizes for priority-aware routers in 4x4 mesh with various traffic patterns have been measured: Reverse (Figure 4.4), Shuffle (Figure 4.5) and Tornado (Figure 4.6).

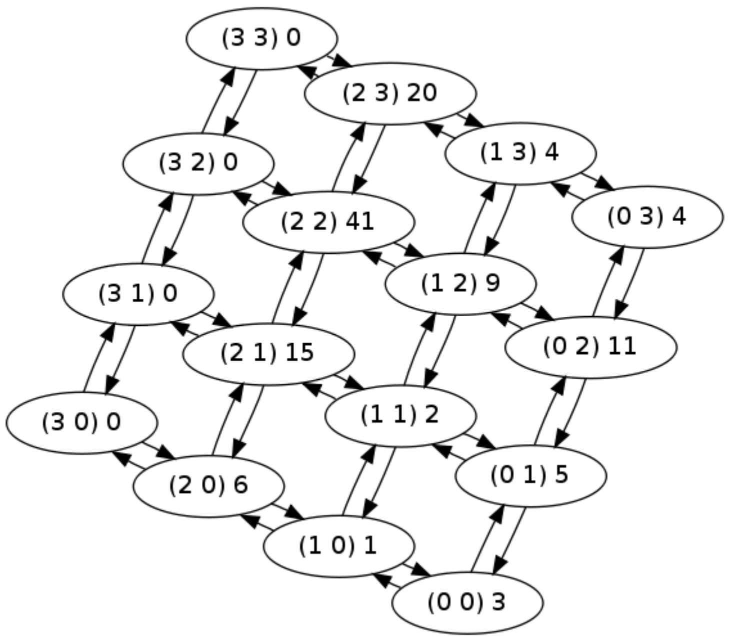


Figure 4.2: Shuffle traffic pattern, XY router - maximum queue length 41

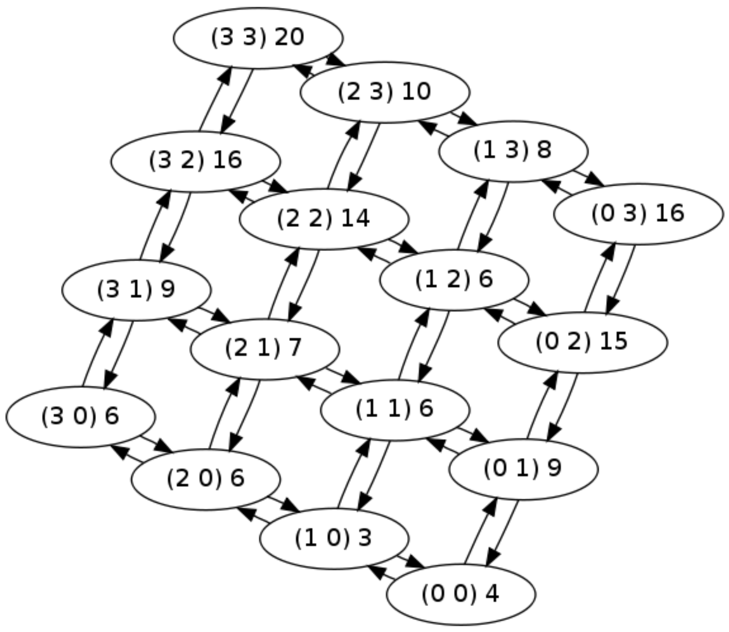


Figure 4.3: Tornado traffic pattern, XY router - maximum queue length 20

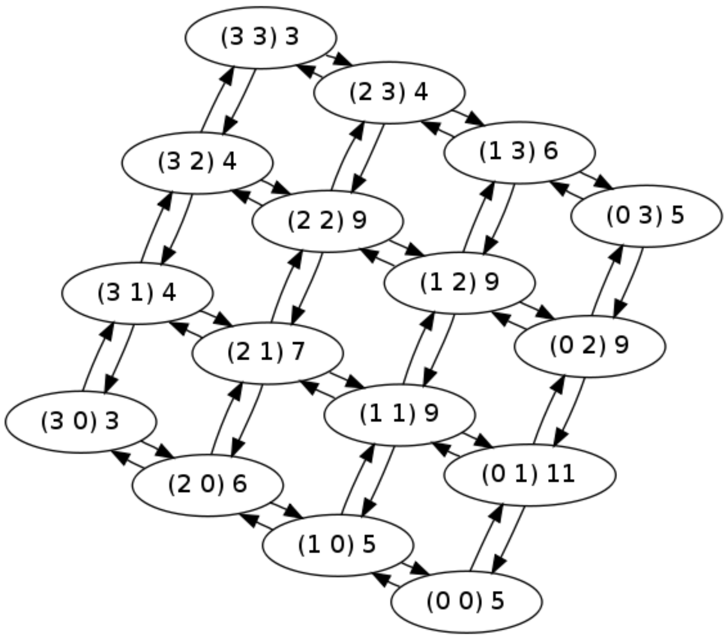


Figure 4.4: Reverse traffic pattern, priority-aware router - maximum queue length 11

Tornado traffic pattern caused the biggest queue sizes in the set of patterns under experiment.

Average of maximum queue lenghts for all traffic patterns in PARouter has been 50 % of that in XYRouter.

Standard deviation of queue sizes has been also calculated for different traffic patterns and was as following: Reverse - 2.44 , Shuffle - 3.52 , Tornado - 3.67 . Tornado also had the greatest standard deviation. Average deviation for PARouter has been 47 % of that in XYRouter.

4.1.3 Impact on average number of hops

Average number of hops is naturally dependent on the traffic pattern which is not a subject of this thesis. The routing algorithm, however, may be most effective in terms of number of hops, for one of the traffic patterns.

Table 4.1: Average number of hops

	Reverse	Shuffle	Tornado
XY routing	4.04	2.71	4.00
Priority-aware	4.09	2.74	4.06
Priority-aware 2	4.10	2.76	4.01

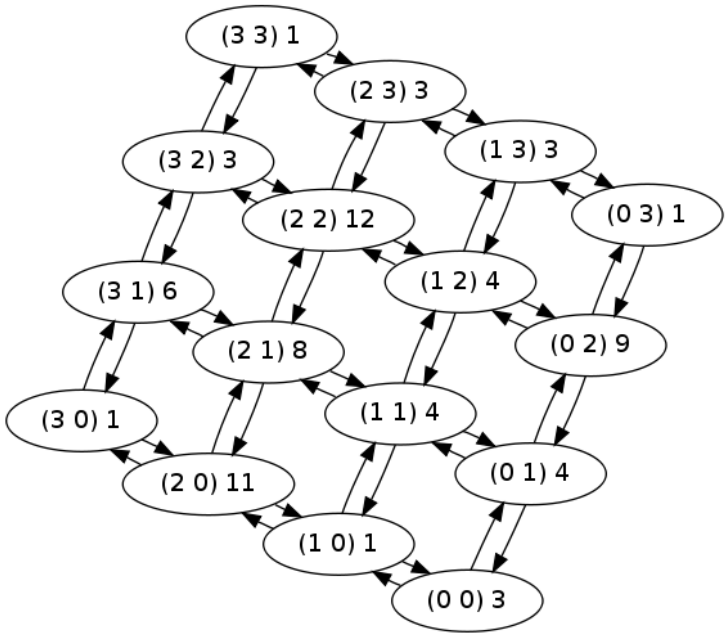


Figure 4.5: Shuffle traffic pattern, priority-aware router - maximum queue length 12

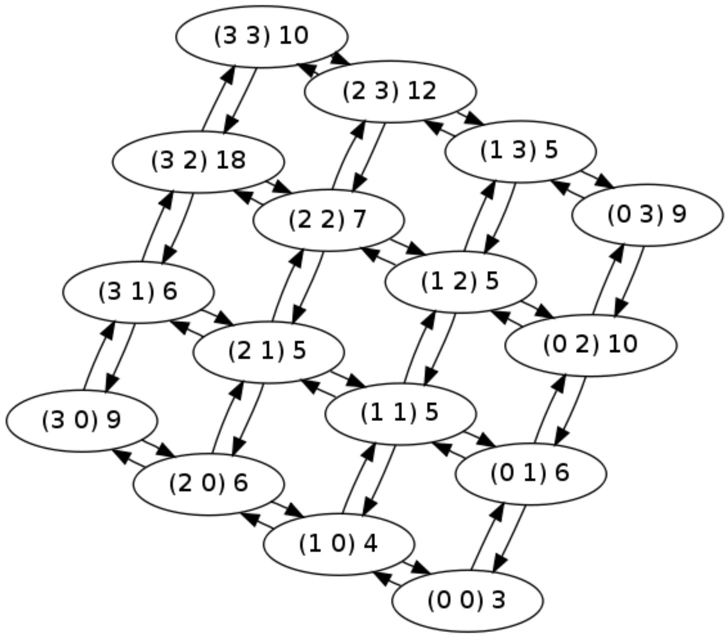


Figure 4.6: Tornado traffic pattern, priority-aware router - maximum queue length 18

4.2 Impact of network load on performance metrics

4.2.1 Experiment conditions

An experiment has been conducted on 4x4 mesh during 10000 simulation ticks. Reverse traffic pattern has been used in the experiment. "Queue near full" threshold has been set at $Q_{FULL} = 8$. Maximum number of "bad hops" has been set at 0 for highest priority, 3 for medium and 5 for lowest.

In order to enforce hot spots in the network, for each generated packet, both coordinates of its source address are chosen using Gauusian random function with $\mu = 2.5$, $\sigma = 0.9$ with saturation function applied afterwards.

Probability of injecting packet has been variable. It is presented as a number between 0 and 1, where 1 means that during every simulation tick, a packet will be injected.

4.2.2 Impact on queue sizes

Minimum required queue size has been measured. This has been achieved by allowing algorithm to have arbitrary queue size and measuring its top size during simulation.

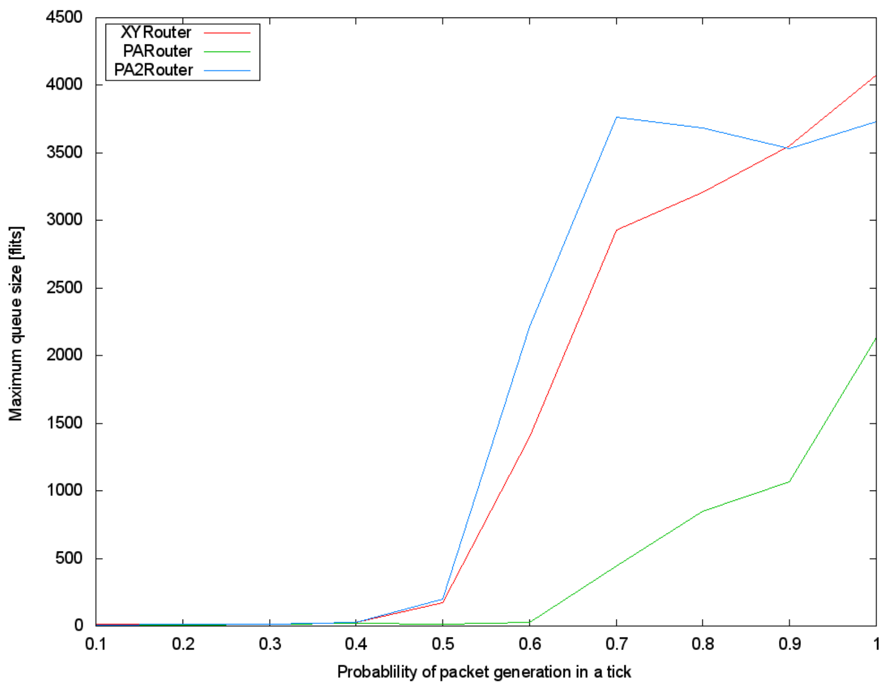


Figure 4.7: Queue sizes as a function of network load

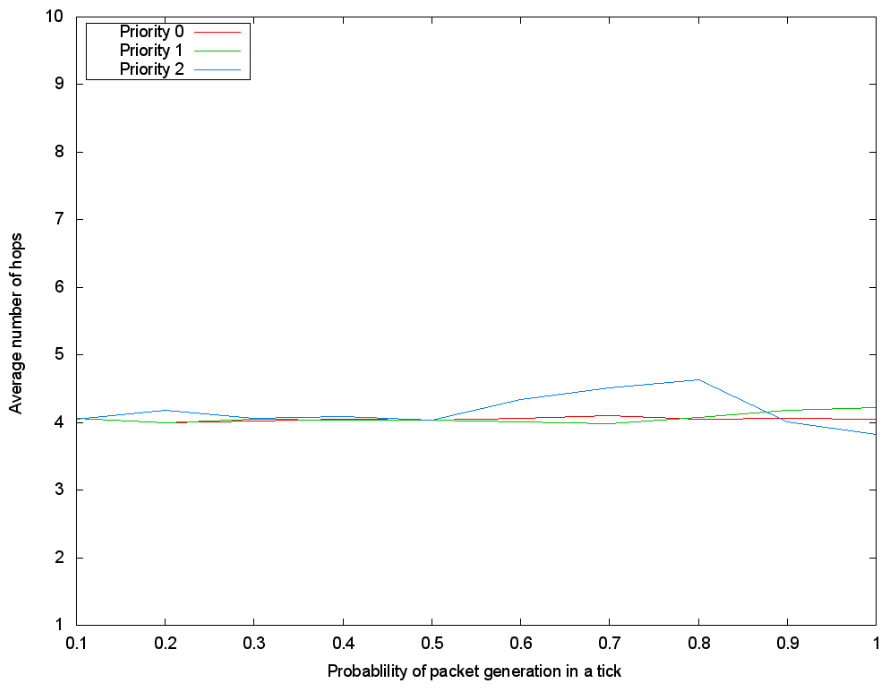


Figure 4.8: Average number of hops in XY routing

Priority-aware routing (PARouter) keeps reasonably small queue size of 25 even at high load (probability of generating packet = 0.6), when XY router would need a queue size of 1552 and PA2Router would need 2211 slots, which is two orders of magnitude higher.

At higher load, queues in PARouter are too big to be implemented in hardware, but growth rate (tangent) is still smaller than that of remaining routing algorithms.

4.2.3 Impact on number of hops

Impact of network load on average number of hops traversed by packets, broken down by priority, has been measured for each of algorithms.

Figure 4.8 shows that the number of hops in XY routing is not dependent on packet priority. This is exactly as expected.

Figure 4.9 shows that in Priority-aware routing, number of hops is the same as in XY routing until queues reach Q_{FULL} value. Then, low priority packets make much more hops while highest-priority ones still do as much hops as in XY routing.

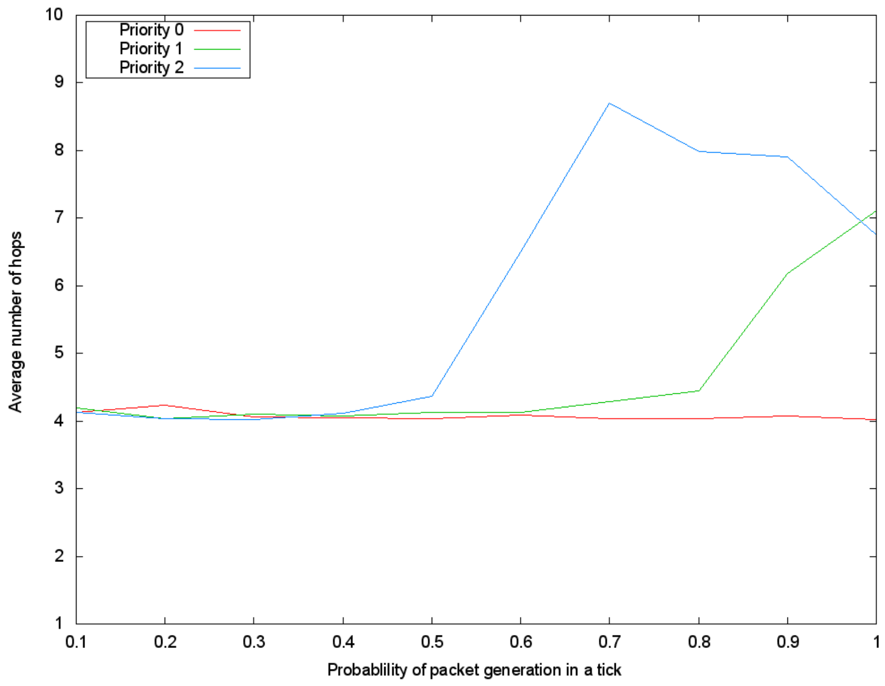


Figure 4.9: Average number of hops in PA routing

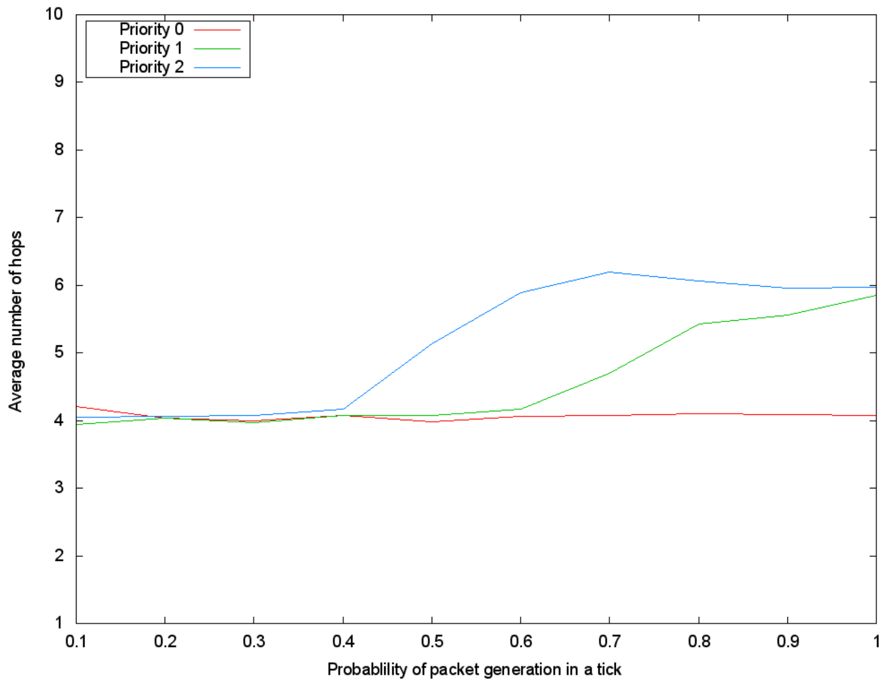


Figure 4.10: Average number of hops in PA2 routing

For instance, at load 0.3 with PARouter, priority 1 packets make on average 101 % of the number of hops priority 0 packets make, and priority 2 packets make 99 %. At a higher load, 0.6, these numbers are respectively 100 % for priority 1 and 158 % for priority 2, and for 0.8 these numbers equal 110 % and 197 %.

Similar thing happens with PA2Router, as seen on figure 4.10: at load 0.3, priority 1 packets make on average 99 % of the number of hops priority 0 packets make, and priority 2 packets make 101 %. At a higher load, 0.6, these numbers are respectively 102 % for priority 1 and 145 % for priority 2, and for 0.8 these numbers equal 132 % and 147 %. While average number of hops in PA2Router is less than in PARouter, the results presented below show that this has little impact on latency.

4.2.4 Impact on latency

Ticks are better measure of routing algorithm performance than hops, because they measure time it actually takes for the packets to get from source to destination router. Unlike hops, it includes waiting in input queues, so the value being measured is packet latency.

Next to throughput and jitter, latency is one of the most important and standard tools of performance evaluation[21].

The average number of ticks for XY router (Figure 4.11) and Priority-aware router (Figure 4.12) is similar, with values a little smaller for PARouter. In PARouter, for load equal to 0.3, packets with priority 1 have on average 131 % of latency of packets with highest priority, and packets with priority 2 have 201 %. The difference is more significant than in number of hops, but still quite small.

For load 0.6, packets with priority 0 still have the same latency as for load 0.3, but the difference between priorities is much more noticable: packets with priority 1 trip 235 % the time of packets with highest priority, and for packets with lowest priority this number is 1693 %. For load 0.8, these percentages are 844 % and 2106 % respectively.

PA2Router (Figure 4.13) performs better in terms of ticks passed than both XY and PA routers. This is especially true for very heavily loaded networks, where packets of lower priority arrive after significantly lower time. For load 0.3, packets with priority 1 have on average 131 % of latency of packets with highest priority, and packets with priority 2 have 201 %. For load 0.6, just as in PARouter, latency of highest priority packets did not increase, but the difference between priorities is higher: packets with priority 1 trip 235 % the time of packets with highest priority, and for packets with lowest priority this

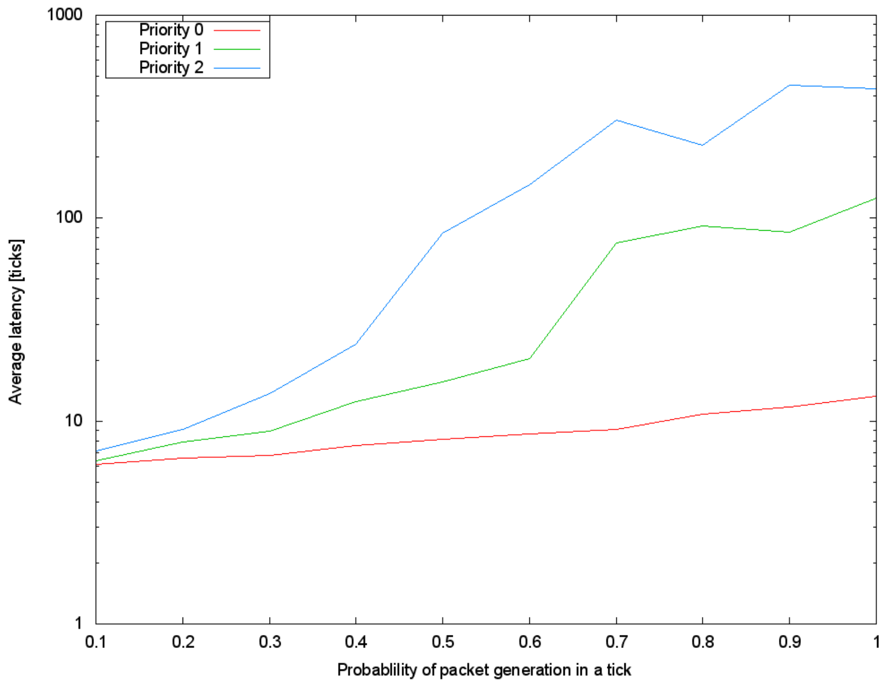


Figure 4.11: Average number of ticks in XY routing

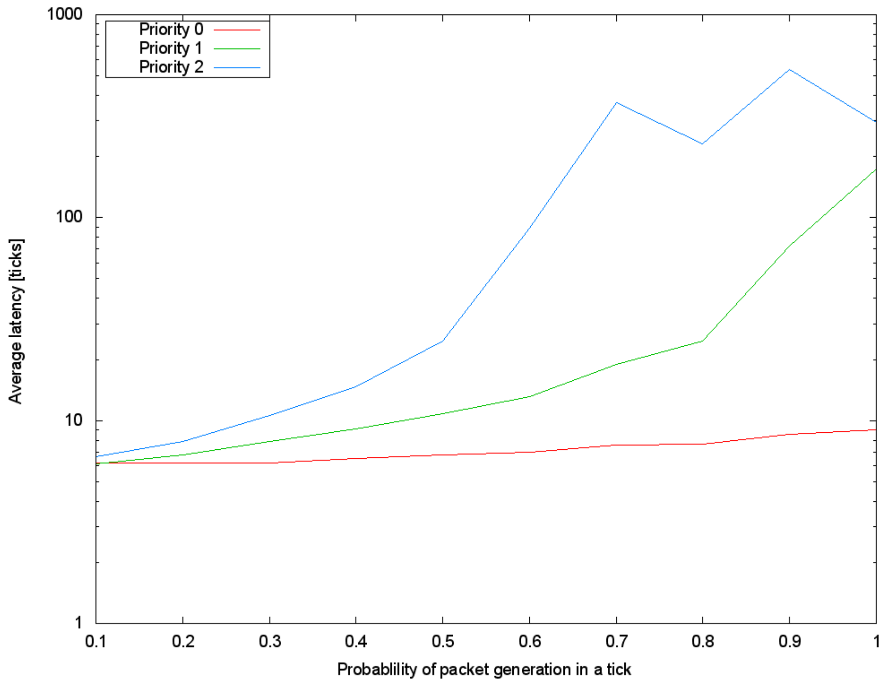


Figure 4.12: Average number of ticks in PA routing

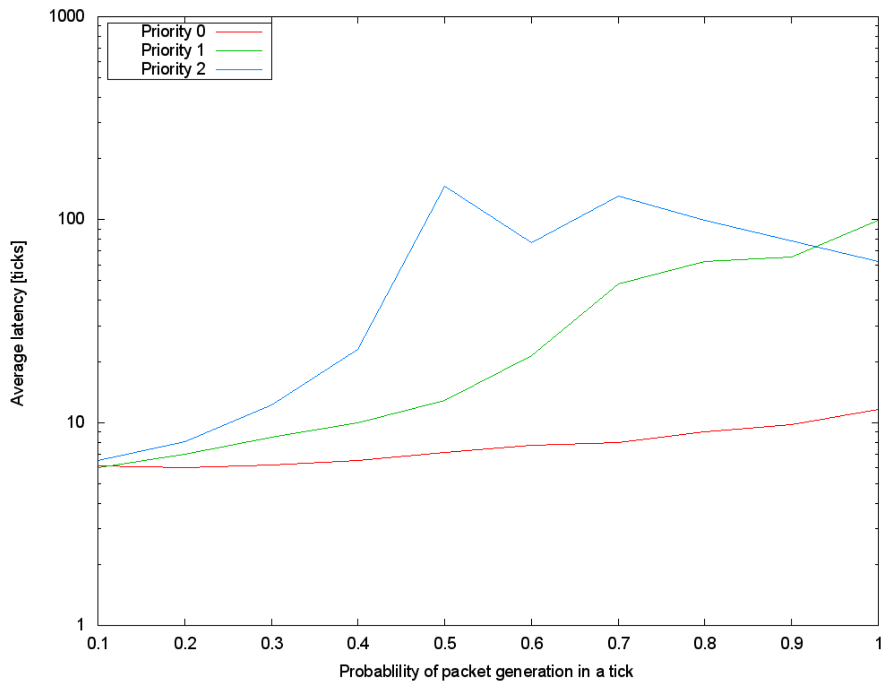


Figure 4.13: Average number of ticks in PA2 routing

number is 1693 %. For load 0.8, these percentages are 844 % and 2106 % respectively.

4.3 Impact of network size on performance metrics

An experiment has been conducted where mesh sizes were changed in range of 3x3 up to 16x16 routers. Impact of network size on performance metrics is an indicator of algorithm scalability - algorithms with polynomial or exponential dependencies are not suitable for practical use.

4.3.1 Experiment conditions

An experiment has been conducted during 10000 simulation ticks. For every combination of router and size, experiment has been conducted five times and averages were reported. Reverse traffic pattern has been used in the experiment. "Queue near full" threshold has been set at $Q_{FULL} = 8$. Maximum number of "bad hops" has been set at 0 for highest priority, 3 for medium and 5 for lowest.

Probability of injecting packet during a simulation cycle has been fixed at

Table 4.2: Maximum size of queues for changing mesh size

Mesh size	XYRouter	PARouter	PA2Router
4x4	1037	41	2257
5x5	808	21	1432
6x6	843	50	1524
7x7	705	31	1386
8x8	743	24	780
9x9	516	24	510
10x10	498	20	71
11x11	485	19	46
12x12	414	17	35
13x13	464	20	34
14x14	516	18	33
15x15	577	16	32
16x16	457	15	34

0.6. Such a high value has been selected because at this load, QFULL signals are emited and performance metrics are different between algorithms (Figure 4.7, Figure 4.9). In order to enforce hot spots in the network, for each generated packet, both coordinates of its source address are chosen using Gaussian random function with $\mu = 2.5$, $\sigma = 0.9$ with saturation function applied afterwards.

Size of 2-mesh has been variable sizing from 3x3 to 16x16, all measured meshes were square (same size in both dimensions).

4.3.2 Impact on queue sizes

Maximum number of flits in queues has been measured for five experiments, averaged and presented in Table 4.2 as well as Figure 4.14.

Both XY and old version of PA router (PA2Router) required non-realistic queue sizes at this load. At network size of 10x10, PA2Router decreased its required queue size by two orders of magnitude, thus making it suitable for practical use. The decrease is caused by a larger number of available alternative routes in a larger network.

4.3.3 Impact on number of hops

Number of hops has been measured for all three routers, broken down by priority.

In XY router (Figure 4.15), number of hops has been close to constant at values less then 5 for sizes under 8x8, and then started growing lineary reaching 20 at size of 16x16.

This result was similar for PA router (Figure 4.16), with similar values. PA2 router (Figure 4.17), while retaining smaller number of hops for smaller meshes, had higher average number of hops for lower priority packets.

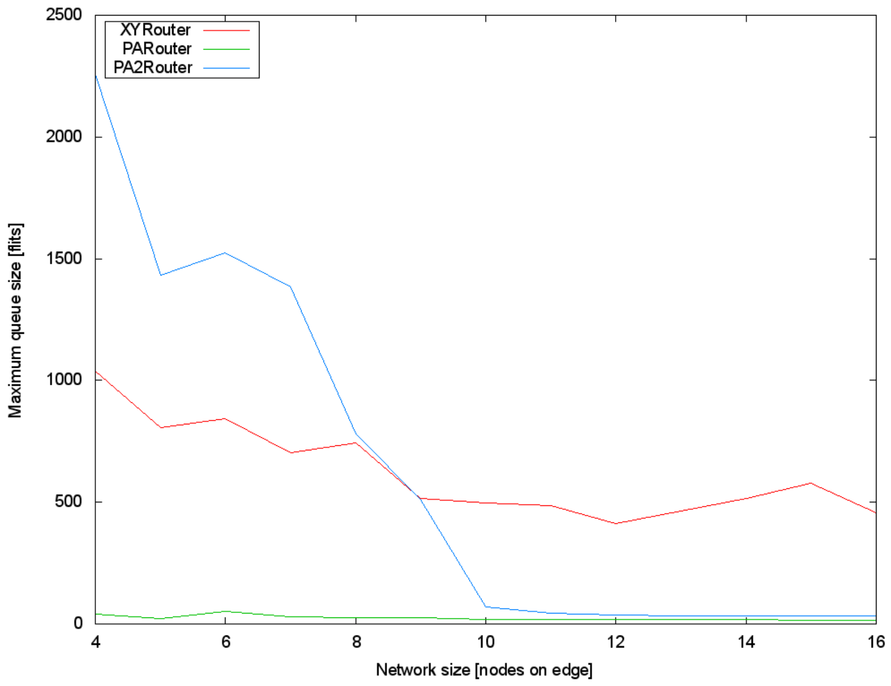


Figure 4.14: Maximum size of queues for changing mesh size

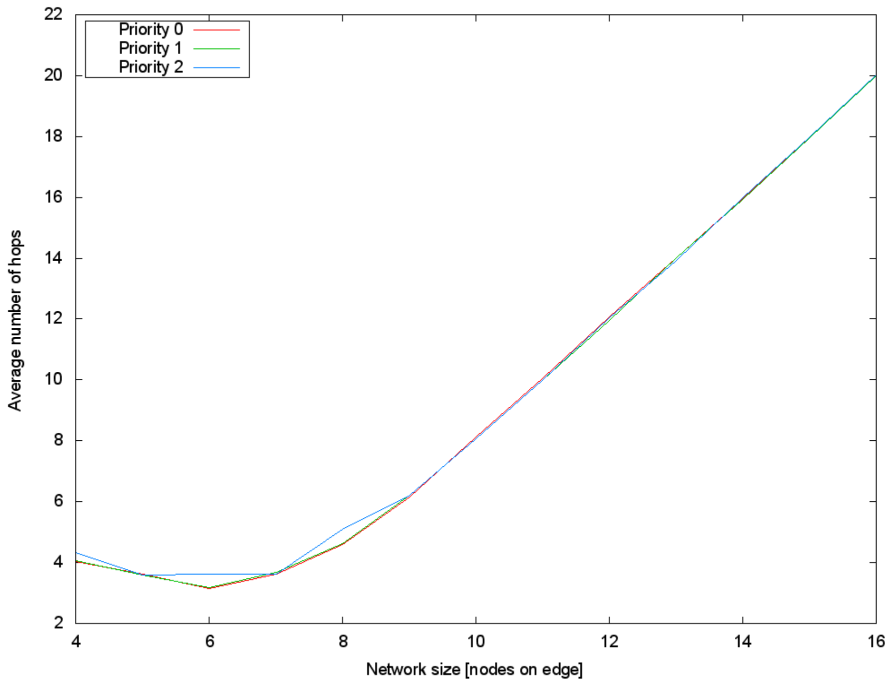


Figure 4.15: Average number of hops for changing mesh size - XY router

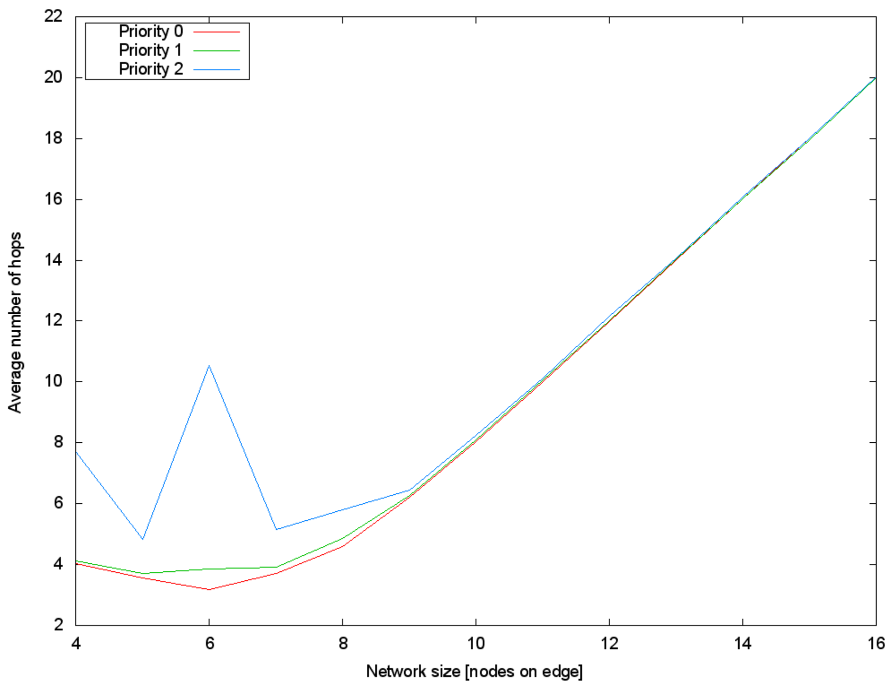


Figure 4.16: Average number of hops for changing mesh size - PA router

In all algorithms, network size does not affect variance of number of hops calculated for all three priorities.

Linear regression of average number of hops (regardless of priority) has been calculated for all three types of routers. Because the growth seems to be linear from 8x8 mesh up, regression has been calculated for size at least 8x8.

For XYRouter, $a = 3.81, b = -10.54$ where a is growth rate ($tg\alpha$) and b is constant factor. For PARouter, $a = 3.20, b = -12.57$ and for PA2Router, $a = 4.40, b = -8.36$.

Slowest growth rate has been achieved by PARouter while PA2Router had the fastest growth rate.

4.3.4 Impact on latency

For all three kinds of routing, despite the large amount of random factor, it is clearly visible that latency increases lineary with mesh growth. This is partially because of traffic pattern, which routes packet on the other end of network, so average distance between source and destination router is in proportion to network size.

XYRouter (Figure 4.18) has the highest growth rate and highest latency for 16x16 network, PA2Router (Figure 4.20) is in the middle and PARouter (Figure

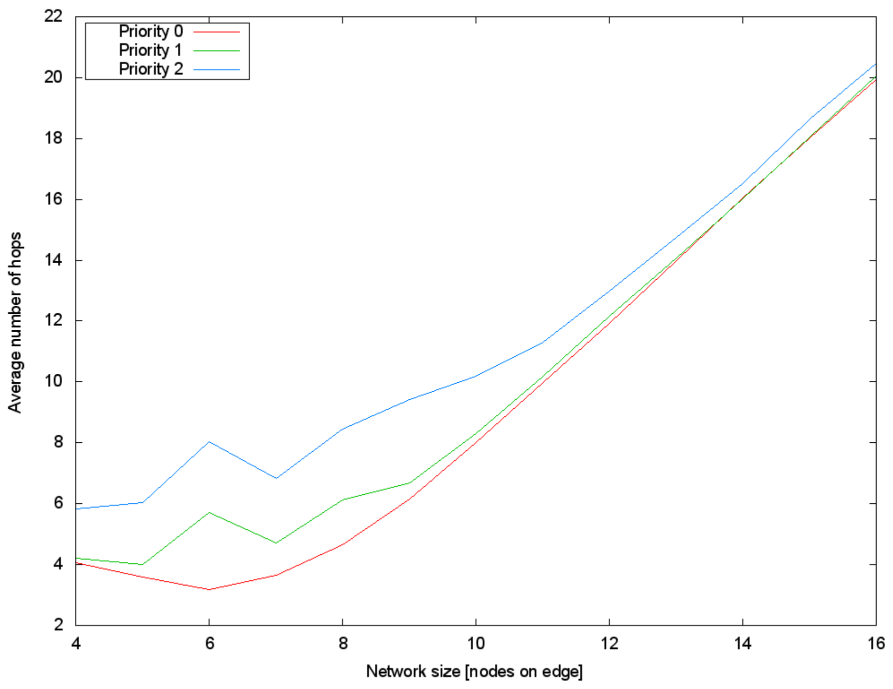


Figure 4.17: Average number of hops for changing mesh size - PA2 router

4.19) has the least growth rate and least latency for largest network.

Linear regression has been calculated for latencies averaged by priority. For XY router, $a = 25.22, b = -128.18$ where a is growth rate $(tg\alpha)$ and b is constant factor. For PA router, $a = 1.24, b = 25.27$ and for PA2 router $a = 2.13, b = 28.26$. It is therefore visible that in PARouter, average latency growth is slowest with mesh size growth, and XYRouter has the highest growth rate. For all three types of routers, dependency is close to linear.

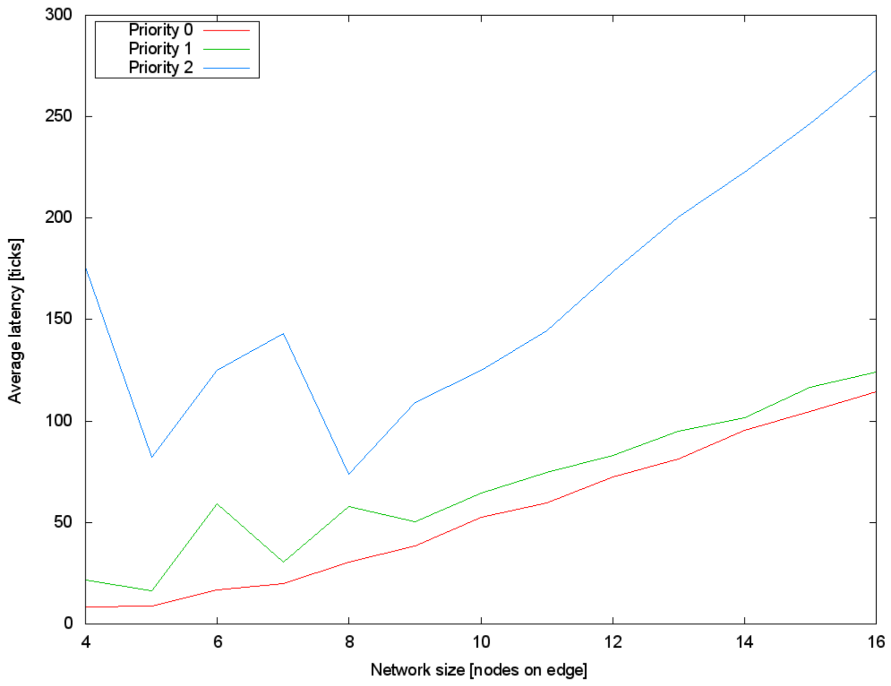


Figure 4.18: Average latency for changing mesh size - XY router

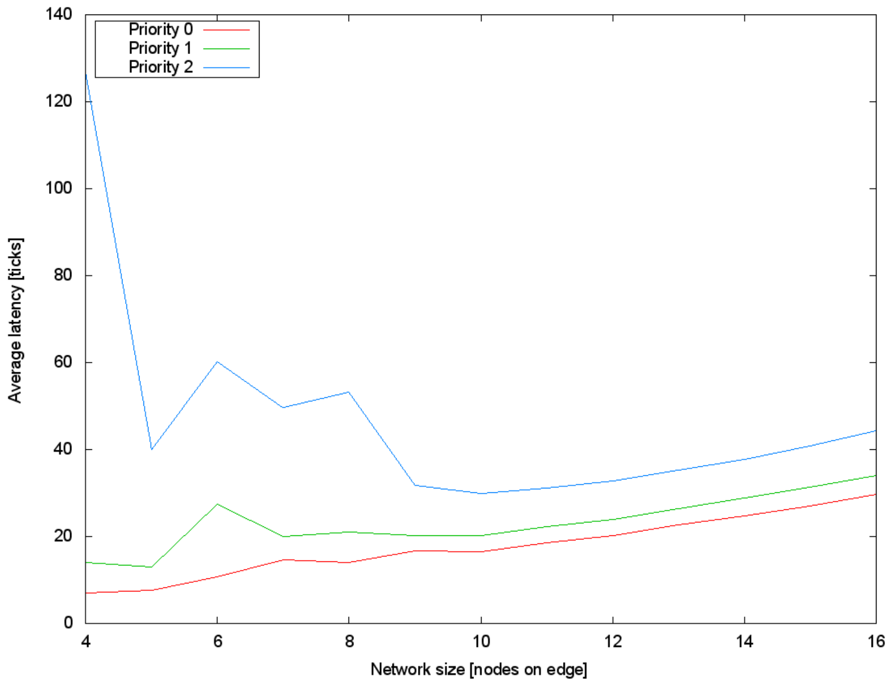


Figure 4.19: Average latency for changing mesh size - PA router

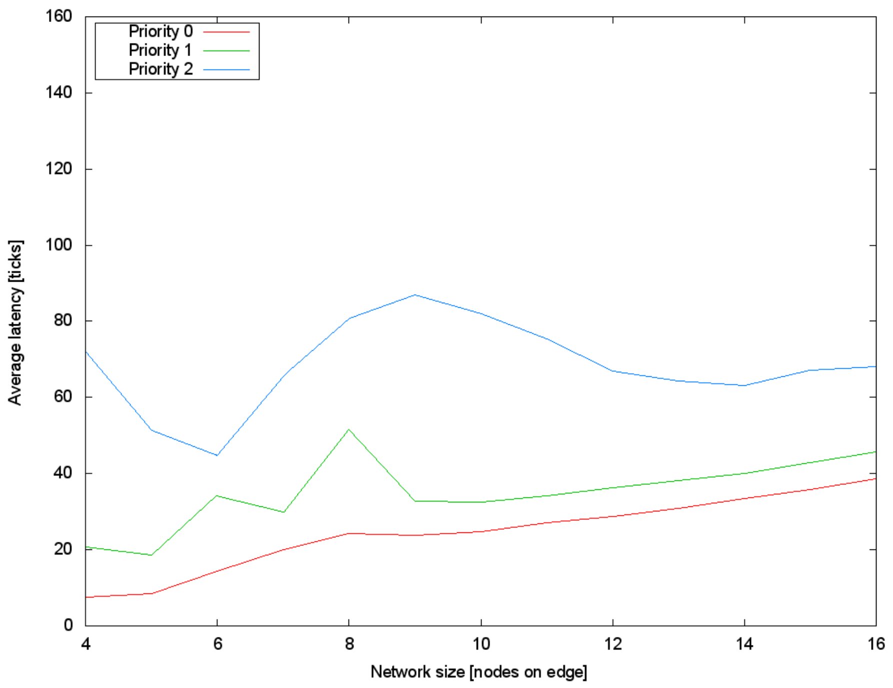


Figure 4.20: Average latency for changing mesh size - PA2 router

Chapter 5

Conclusion

Authors of literature describing state of the art in Network on Chip technology agree about the need for Quality of Service in these networks in order to enable better resource utilization [17] [20]. Many solutions for QNoC have been already proposed, but none of the ones evaluated offered elimination of congestion using different routes for different packet priorities.

The proposed algorithm, priority-aware routing, is capable of supporting networks utilizing Quality of Service by decreasing network congestion, while keeping minimal routes for high-priority packets. It is a local adaptative algorithm that at every hop of route checks if there is a non-overloaded router in the next hop of minimal route. If no, it redirects low-priority packets to a non-minimal route, to avoid congestion. Every packet keeps misroute count to prevent livelocks, and maximum value of this count is dependent on the priority. The routing algorithm should be used in conjunction with a priority-aware scheduling algorithm.

Compared to traditionally used XY routing, priority-aware routing requires smaller queues for the same traffic, or supports larger traffic at the same queue size, at a price of some communication overhead. It needs one binary control signal and 2 to 5 extra bits in a packet. Lower priority packets make a larger numer of hops and thus have higher latency. This effect may be increased when an algorithm utilizing QoS priorities heavily is utilized too. The latency of high-priority packets is comparable to XY.

With increasing mesh size, queue sizes don't change, and latency grows lineary, therefore the algorithm is scalable to large networks.

Research should be done on measuring influence of Q_{FULL} factor and number of "bad" hops allowed for different priorities. It could be checked if these values could be chosen in an adaptative manner.

The algorithm includes considerably more logic than standard XY algorithm.

Whether benefits of using this algorithm compensate logic and communication overhead is dependent of particular application and to be investigated by designer.

List of Figures

List of Tables

Bibliography

[1] G. Moore, Cramming more components onto integrated circuits, 1965.

[2] R. Marculescu, Ü. Y. Ogras, L.-S. Peh, N. D. E. Jerger, and Y. V. Hoskote, IEEE Trans. on CAD of Integrated Circuits and Systems **28**, 3 (2009).

[3] M. Mitic and M. Stojcev, A survey of three system-on-chip buses: Amba, coreconnect and wishbone, 2006.

[4] T. Bjerregaard and S. Mahadevan, ACM Comput. Surv. **38** (2006).

[5] V. Zdornov, Adaptive routing in (q)noc, 2008.

[6] P. Vellanki, N. Banerjee, and K. S. Chatha, Integration **38**, 353 (2005).

[7] A. Hansson, M. Coenen, and K. Goossens, Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip, in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pp. 954–959, San Jose, CA, USA, 2007, EDA Consortium.

[8] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003).

[9] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, Journal of Systems Architecture **50**, 105 (2004).

[10] J. Delorme, An automatic design flow for mapping application onto a 2d mesh noc architecture., in *PATMOS*, edited by N. Azmard and L. J. Svensson, , Lecture Notes in Computer Science Vol. 4644, pp. 31–42, Springer, 2007.

[11] D. Atienza *et al.*, Integration **41**, 340 (2008).

[12] S. Kumar *et al.*, VLSI, IEEE Computer Society Annual Symposium on VLSI **0**, 0117 (2002).

[13] R. Holsmark, M. Palesi, and S. Kumar, J. Syst. Archit. **54**, 427 (2008).

[14] R. Gindin, Architecture and routing for noc-based fpga, 2007.

[15] P. Dziurzanski and T. Maka, Heuristics core mapping in on-chip networks for parallel stream-based applications, in *ICCS '08: Proceedings of the 8th international conference on Computational Science, Part I*, pp. 427–435, Berlin, Heidelberg, 2008, Springer-Verlag.

[16] D. Andreasson and S. Kumar, Slack-time aware routing in noc systems, in *ISCAS (3)*, pp. 2353–2356, IEEE, 2005.

[17] K. Goossens, J. Dielissen, and A. Rădulescu, IEEE Design and Test of Computers **22**, 414 (2005).

[18] F. Steenhof, H. Duque, B. Nilsson, K. Goossens, and R. Llopis, Design, Automation and Test in Europe Conference and Exhibition **2**, 29 (2006).

[19] T. Marescaux and H. Corporaal, Introducing the supergt network-on-chip: Supergt qos: more than just gt, in *DAC '07: Proceedings of the 44th annual conference on Design automation*, pp. 116–121, New York, NY, USA, 2007, ACM.

[20] M. A. Al Faruque and J. Henkel, Int. J. Parallel Program. **36**, 114 (2008).

[21] L. Tedesco, A. Mello, L. Giacomet, N. Calazans, and F. Moraes, Application driven traffic modeling for nocs, in *SBCCI '06: Proceedings of the 19th annual symposium on Integrated circuits and systems design*, pp. 62–67, New York, NY, USA, 2006, ACM.