Evaluation of a Large-Scale SpiNNaker System

Javier Navaridas[1], Mikel Lujan[2], Jose Miguel-Alonso[1], Luis Plana[2] and Steve Furber[2]

Abstract — **The SpiNNaker system is a novel SoC design that aims to provide real-time simulation of very large-scale neural networks, focusing on biologically inspired spiking neuron networks. In this document we describe its architecture, with focus on the interconnection network. We aim to the packet dropping mechanisms that provide deadlock and live-lock avoidance. The first one is based on the time a router keeps a packet before dropping it when unable to forward it. The second one is a mechanism based on global ages that drops a packet that has not been delivered after two ages. This work evaluates the largest configuration of the system with different values for the waiting time to avoid deadlock. Furthermore, taking into account the maximum observed latencies, we propose an age length to avoid live-lock. We also select the best performing value of the waiting time, considering the number of packets dropped at different injection rates. The obtained results show that high values of waiting time help reducing the amount of dropped packets when the injection rates are low; in contrast, it is better to keep this value close to zero when injection rates are close to saturation. System performance under different failure scenarios is also tested, and results show that, with failures, the effects of saturation appear at lower injection rates.**

Keywords — **Interconnection Networks, Parallel systems, Performance evaluation, Real-time applications, Systems on Chip.**

I. Introduction

THE SpiNNaker Massively Parallel Neural Networks Simulator has been designed to mimic neural activity [3], which is characterized by massively parallel processing and high levels of communication between processing units [1]. The system is highly scalable and at the largest configuration, with more than one million processors, will support real-time simulations with over one billion neurons. One possible use of this system, yet not the only one, is to be the mind of a robotic system providing real-time stimulus-response behaviour [2].

This paper is focused on the interconnection network that will support inter-processor communication to simulate synaptic connections. We will explore several parameters of the routers that compose the network, in order to find those values that provide the best system performance. The figures of merit will be the latency-related characteristics of the system as well as the number of packets dropped by the network due to congestion. Furthermore the system performance will be tested under scenarios in which the network suffers of broken links, in order to assess the robustness of the system against failures.

Note that the SpiNNaker system is rather different from the typical architecture of regular High Performance Computing systems. HPC systems are commonly built with very fast CPUs over not-so-fast networks; in contrast, the SpiNNaker system is built with low-consumption slow CPUs working at 100MHz (2 orders of magnitude below regular HPC systems) and an over-dimensioned network that is able to communicate at 1 Gbps (one order of magnitude below current HPC networks). For this reason the kind of evaluation we will carry out on this network will not be the same we would do on a typical HPC interconnection system.

The rest of this paper is organized as follows. Section II describes in depth the SpiNNaker system architecture. In Section III the models of the components in the simulation-based environment are shown. The results of the experimental work are presented and discussed in Section IV. We close this paper in Section V with the conclusions of this evaluation work, and an outlook of further research activities.

II. SpiNNaker Architecture

SpiNNaker is a system on chip (SoC)-based architecture designed to support parallel distributed computing with high bandwidth and low-delay communications. In this section we will perform a brief description of its architecture – we encourage the interested reader to look at [6] for a more detailed description at hardware level. Moreover an overview of neural networks applications and systems from an engineering point of view is shown in [4].

The basic SpiNNaker chip comprises 20 embedded ARM968S-E processing cores, each of them with a tightly-coupled dedicated memory that can hold 32KB of instructions and 64KB of data. Processing units are also provided with other useful modules such as a Timer, Vector Interrupt Controller (VIC),

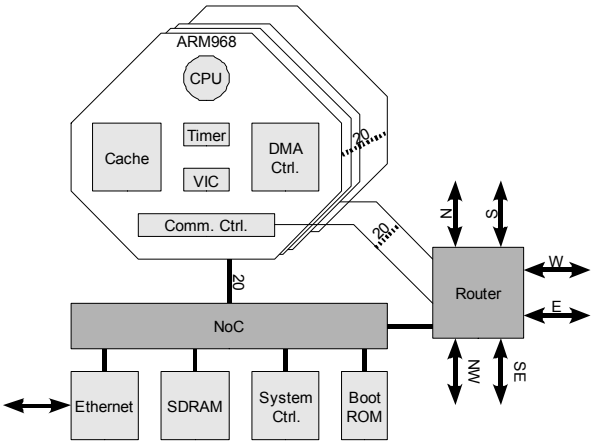

Fig. 1. Schematic model of the SpiNNaker chip with all its components depicted.

[1] Authors are with the Dept. of Computer Architecture and Technology , The University of the Basque Country, San Sebastian, Spain, (contact e-mail: *javier.navaridas@ehu.es*).

[2] Authors are with The Advanced Processor Technologies Group, The University of Manchester, Manchester, United Kingdom.

Communication Controller and DMA Controller. The architecture of the SpiNNaker chip is depicted in Fig. 1. Note that router architecture will be detailed in the following paragraphs.

All the cores in a chip share a SDRAM of up to 128 MBytes in which synaptic connection information is stored. Access to this shared storage space is carried out by means of an asynchronous Network-on-Chip (NoC) [8]. This network is called the System NoC and provides a bandwidth of 8 Gbps. It is also used to connect other resources in the chip, as for example the Boot ROM, the System Controller and the Router – note that the router will be accessed through the NoC just for configuration purposes, during regular utilization the ARM cores will use the communication controller to send or receive packets. We would like to remark that this NoC provides higher communication bandwidth, lower contention and lower consumption than any typical bus architecture [7].

Each chip also has a built-in Ethernet sub-module connected to the NoC. It will support communication with external systems using the common TCP/IP protocol stack. The Ethernet interfaces can also be used to increase redundancy, and thus fault-tolerance, when utilized to re-connect areas of the network that are split due to system failures. Note that although all the chips are provided with that Ethernet connection, only a few will make use of it, in order to reduce power consumption and minimize the computing resources needed to implement the protocol stack – note that one of the cores will have to be in charge of the Ethernet connection and protocols when it is in use.

SpiNNaker chips are arranged in a 2D mesh topology with links to the neighbours in North, South, East, West, South-East and North-West. An 8x8 instance of this topology is depicted in Fig. 2. Note that chips at the network boundaries are connected by means of peripheral, wrap-around links that are not shown in the figure for the sake of simplicity.

As stated before, each chip incorporates a router that

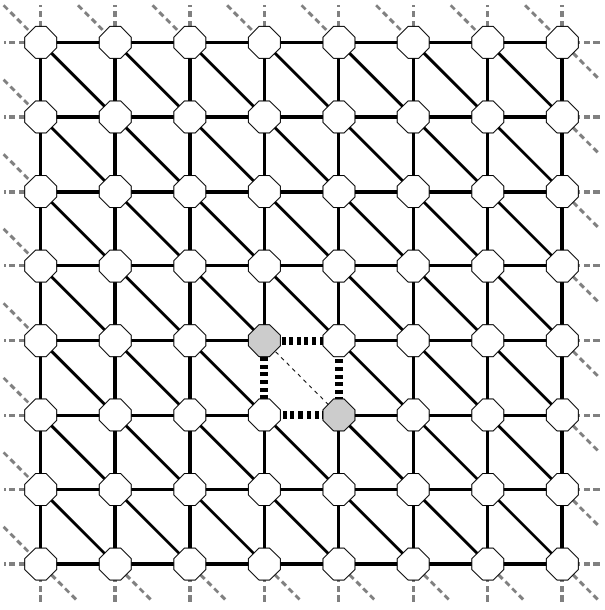

Fig. 2. Example of an 8x8 SpiNNaker topology. Peripheral connections are not depicted for the sake of clarity. The regular route (thin and slashed line) and the two emergency routes (thick and dotted lines) between the two shaded nodes are shown.

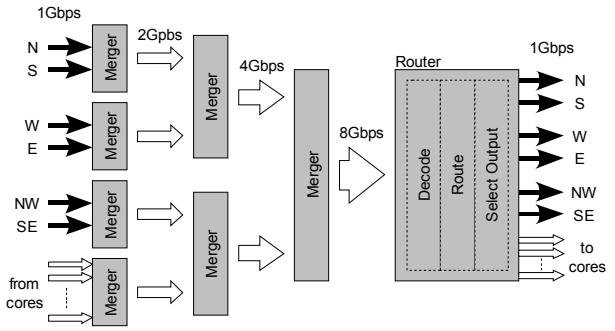

Fig. 3. Router Architecture. Black arrows represent links outwards the chip. White arrows represent hard-wired links within the chip.

allows inter-chip communication. A depiction of its architecture is shown in Fig. 3. It has 20 ports for internal use of the embedded cores and six ports to communicate with six adjacent chips. All ports are full-duplex and implement asynchronous protocols. The organization within the router is hierarchical; ports are merged in three stages before using the actual routing engine. Note that a router is able to forward a single packet at once, but it works faster than the transmission ports. Thus, most of the time the router will be idle, and the router delay does not affect the pace at which packets are processed.

The router is designed to support point-to-point as well as multicast communications. Information interchange is performed using small, 40-bit packets. It is important to indicate that routers make routing decisions based on the source address of the packets, i.e. packets do not contain information about the destination node(s), but only about the neuron that has been fired, being the network itself which will deliver the packets to all chips containing neurons that have synaptic connections with the source neuron. This information is embedded in the 1024-word routing tables available at the routers. To assure that the system works properly with such a reduced number of entries in the routing tables, the routers are designed to perform a default routing that sends the packet to the port opposite to the one the packet comes from; for example, if the packet comes from the North it will be send to the South. This default routing will be performed when there is no entry in the table for the source chip. The expected shape of the routes between every two chips is by means of two straight lines and will only need the inflection point [5] (the point with these two lines are joined) to be stored in the routing tables. It is remarkable that the network topology allows two-hop routes to go from a chip to each one of its neighbours, see again Fig. 2. These two-hop paths between neighbour nodes are denoted as emergency routes and may be invoked to handle transient congestion states as well as with link failures in order to bypass problematic links.

The flow control of this network is very simple. When a packet arrives to an input port, an output port is selected and the router tries to transmit the packet through it. If, after a given amount of time, the packet has not been forwarded, the router will try the emergency routes. Finally, if these are not available, the packet will be dropped to avoid dead-lock. Moreover, in order to avoid live-lock situations, packets have an *age*

field in their header; if two ages pass and the packet has not been delivered, it will be considered as *outdated* and will be dropped. Note that the ages are global to the whole system and have arbitrary length. As a contribution of this work, we provide bounds of the value for this network parameter.

It is remarkable that, emulating the behaviour of actual neural networks, dropped packets are not re-sent, Packet loss is not a key issue if the dropping level is kept low. Actual synaptic signals between neurons in living beings can be lost due to different phenomena, but their nervous systems continue working properly. When the degree of lost packets/signals exceeds regular working levels, the neural network will not work properly regardless of being a living or simulated one. This is the reason to allow some degree of packet dropping, but being concerned about keeping it under control.

III. Experimental Set-Up

A detailed model of the SpiNNaker network has been implemented in our in-house developed simulator INSEE [9]. It contains most of the features of the router and also the topological description of the system. However, in order to be able to confront simulations of large-scale systems, some implementation decisions have been taken. INSEE is a time-driven simulator and that approach has been kept. We model a cycle as the time to route and forward a packet. As described in the previous section, routing is faster than transmission, so we allow the router to process up to six packets in a single cycle, provided that all the involved input and output ports are different. Regarding the routing tables, in the actual system they will be configured in an application basis, thus the table-based routing will not be used in this study; note that this also reduces the computing resources required to perform simulations. As the regular routes between chips in the actual system will attempt to use a minimal path with a single inflection point, we send the packets through a minimal route using Dimension Order Routing (DOR) which emulates the behaviour of actual communications in the SpiNNaker system. Note that the diagonal links will be considered a third dimension (Z) when applying DOR, thus the routes followed by packets will always be (X, Y), (X, Z) or (Y,Z) – a path (X, Y, Z) is not a minimal path. Furthermore, as all the ports from the CPUs inside a chip are merged, we will model all of them as a single injection queue.

We have selected the largest configuration of the SpiNNaker system, which is composed by 64K nodes arranged in a 256x256 2D mesh. This system will be evaluated under point-to-point traffic, *i.e.* the multi-cast engine will not be used. The nodes will be modelled as independent traffic sources that inject packets following a Poisson distribution, in which the injection rate (packets per cycle per node) can be tuned to any desired value. We will evaluate the system under a wide range of injection rates going from 0.001 packets/cycle/node to 0.09 packets/cycle/node, which roughly represent 1.6% and 144% of the system theoretical throughput under uniform traffic. This theoretical value is 1/16, but its computation is outside of the scope of this paper. This will allow us to have a picture of the behaviour of the system under different levels of communication requirements. Note that, in the actual system, the network is expected to run at 10-20% of its maximum capacity, but this wide range of injection rates allows the evaluation of the system even under peaks of network utilization. In this study we will consider a uniform distribution of packet destinations, although it is expected that in the actual system the mapping of the neurons will be optimized in such a way that the communicating neurons are in close proximity. Each node has an injection queue with room for 4 packets. If this queue is full and the node tries to inject a packet, the packet will be dropped because of the lack of room to store it.

Different values of the time to wait before dropping a packet to avoid deadlock, from now on *waiting tim*e, will be tested in order to elucidate an optimal value to be used in the actual system. The values in our set of experiments are 0, 1, 2, 4 and 8 network cycles, and will be denoted as wait=0, wait=1, wait=2, wait=4 and wait=8 respectively. Note that zero-waiting means that if a packet can not be transmitted, it will try the two emergency routes and, if none of them are available, the packet will be dropped immediately. In the rest of the cases, the emergency routes will be tested in the last cycle, just before dropping the packet.

The figures of merit will be the amount of packets dropped with the different injection rates, as well as average and maximum packet latency. Note that maximum latency figures will help to select a good value for the age-based packet dropping mechanism to avoid live-lock in the actual system. Of course, the lower the number of dropped packets, the better the configuration. Analogously, the lower the latency figures, the better the configuration.

Moreover, we will test the system under different degrees of network failure to test the robustness of the design. In this work we will focus only on link failures. Systems with up to 4096 uniformly random link failures (roughly 2% of the system) will be evaluated in order to assess the fault-tolerance capabilities of the system. Low levels of system performance degradation are expected, even in the worst scenarios of system failure. Note that the random link failure may lead to nodes being completely disconnected from the system, with the corresponding loss of all the packets departing from or arriving to this node, as well as all the nodes that have to travel through it. This issue will be taken into account in the actual SpiNNaker system: if a node is disconnected to the rest of the system, it will be considered as a non-working node and will not take part in the neural simulation.

IV. Experimental Results

The results of our experimental work are depicted in Fig. 4. Graphs on the left-hand side show the dropped packet ratio for different values of waiting time and levels of failure of the network. The X axis shows the injection rate normalized to the theoretical throughput. Note that this can be over 100% because the bandwidth of the injection links is greater than the theoretical throughput of the network. In the Y axis the dropped packets are presented, normalized to the number of

injected packets. Note that, in this case, values will be strictly in [0, 1] because it cannot happen that there are more packets dropped than injected, and obviously the number of dropped packets can not be negative.

The graphs on the right-hand side of Fig. 4 show the latency-related figures for different values of the waiting time and levels of failure of the network. X axis is again the injection rate normalized to the theoretical throughput and Y axis shows the delay in terms of network cycles.

Looking at these graphs and focusing on the performance of the 0-failures network, it is clear that the system performs correctly until it reaches saturation at around 80% of the (theoretical) network capacity: the number of packets dropped is very low (when not zero), and the latency is well bounded. The only exception appears when using zero waiting time, a very aggressive policy. In this case, the packet dropping reaches noticeable values at injected loads around 66%.

Once the network is saturated it is obvious that the latency depends greatly on the waiting time. The latency-related figures are better for the lower waiting time policies. The graphs of dropped packets also show that, once the network reaches saturation, high waiting times are counterproductive because they keep packets on the network for too long, a situation that will cause the drop of newly injected packets.

For this set of experiments the waiting time equal to one has shown to be the best performer, because it is the one with the lowest degrees of packet dropping – both below and over the saturation point – and also keeps low the latency-related figures. Higher waiting times do not seem to be a good idea because, although they perform slightly better when the network is not saturated,

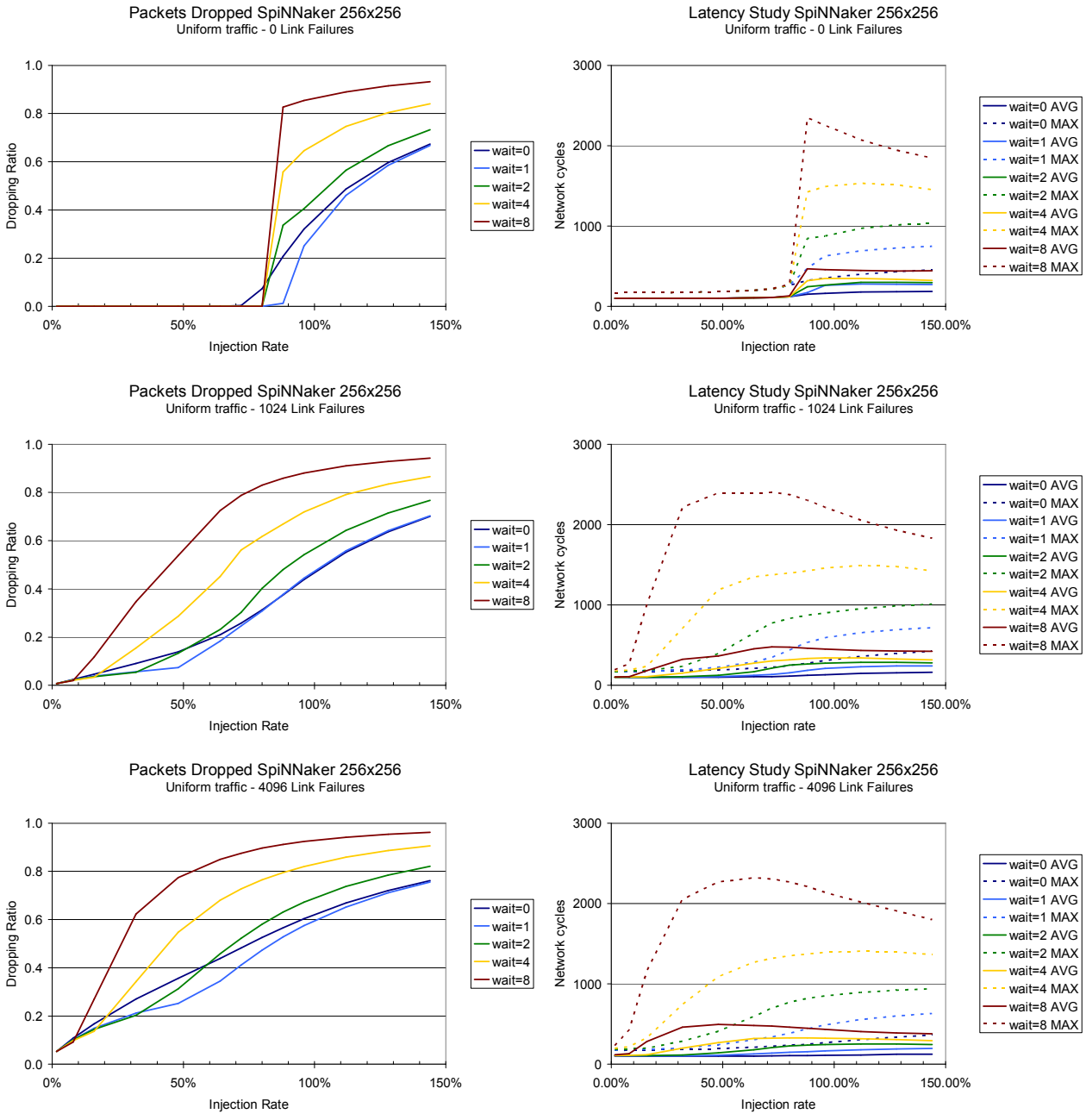

Fig. 4. Results of the experimental work. Left: packet dropping ratio for different degrees of network failure. Right: average and maximum delay in the same scenarios.

transient saturation states may lead to an excessive level of packet dropping and also to latencies that could surpass the constraints of neural network applications.

Regarding the experiments for scenarios with failures, we can see that the apparition of faults result in reaching saturation prematurely. Again, the longer the waiting time, the faster the network reaches saturation states. For example, in the dropped packets graph for the 1024-failures network, the wait=1 configuration performs well until the network capacity reaches around 50%; beyond that point packet dropping figures explode. In the case of wait=2 the inflection point comes at around 35%. For wait=4 and wait=8 the measured figures are on saturation most of the time. However, it is worth to signal that, for the smallest measured values, the wait=8 is the one that drops fewer packets. Regardless of that, again the wait=1 has shown to be the best overall performer, both in terms of dropped packets and latency figures. For the low injection rates in which the system is expected to work, wait=2 and wait=4 lead to slightly lower levels of packet dropping.

In the case of 4096 failure links, the degradation of the network is even more noticeable – note that 2% of the links broken means a severely malfunctioning system. The saturation points for all the cases are lower than in the previous scenario. Again, the overall best configuration is shown to be wait=1, but when the injection rate is low both wait=4 an wait=8 drop a slightly smaller amount of packets. This could be of interest because, as stated before, the interconnection network is designed to run at low utilization levels.

Latency figures are also of interest. Note that when high injection rates lead the network to heavy saturation, the ratio of dropped packet soar, but the latency figures drop. We have to find the explanation for this somewhat unexpected result in the way packets are dropped: usually, most *survivor* packets are those travelling only a few hops, because they stay for less time in the network and, therefore, the probability of going through problematic areas is lower. So what we see is actually a reduction on the average distance of delivered (non-dropped) packets. The simulator reports that, below saturation, average distance traversed by packets is around 100 hops. In contrast, for highly congested situations this distance may fall to values below 50. As the generation of packet destinations does not change, we have to infer that, in saturation, packets travelling far away have serious trouble reaching their destinations.

The results of our experiments may help us to select the length of an *age* in terms of cycles. This will be useful to have a good live-lock avoidance mechanism. Note that, as explained before, all packets that are in the network for more than two ages will be dropped. A length that allows dropping outdated packets as soon as possible, but without dropping slowly-advancing, useful packets is desirable. Age duration could be fixed to the maximum latency value obtained via simulation, which depends on the waiting time. Note that, as ages are global to the whole network, a packet that is injected in the last cycle of an age will be tagged with that age and, therefore, is under the risk of being dropped as soon as the next age finishes, so it will only have one age length plus one cycle to be delivered. Selecting a lower value as age length may lead to unnecessary packet dropping. For example, in the case of wait=1, an age length of 750 cycles would be a good choice. However, note that in this case an outdated packet may wander around the network for up to 1500 cycles. Table I summarizes the maximum latencies measured, for the different waiting time values and levels of network failures.

TABLE I.

MAXIMUM LATENCIES MEASURED FOR DIFFERENT VALUES OF THE WAITING TIME AND DIFFERENT DEGREES OF NETWORK FAILURE.

	wait=0	wait=1	wait=2	wait=4	wait=8
0 faults	455	749	1035	1534	2346
1024 faults	425	717	1009	1492	2403
4096 faults	364	631	939	1406	2319

Regarding the robustness of the system, we can see that the SpiNNaker network manages to work properly with 1024 failures (about a 0.5% of the network) when the injection rate is not very high. With wait=1 and at 50% of the network capacity, less than 7.5% of packets are dropped. In the case of 4096 failures the system still manages to work, but dropping 1 of each 4 packets in the best configuration at 50% injection rate. However note that this study has been performed with dimension order routing, a mechanism that is unaware of system failures, while the actual SpiNNaker routers would use routing tables filled *ad-hoc* for each application running on the system. This table-filling process would be aware of network failures, thus avoiding the non-operational areas of the network [5].

V. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed the design and architecture of the SpiNNaker system. This system will be used to perform real-time simulation of spiking neural networks. Some goals built-in into the design are low power consumption and high tolerance to failures.
In order to be a robust system the SpiNNaker architecture relies on redundancy both in terms of computing and communicating elements.

A performance evaluation of the system focusing on the interconnection network has been carried out, via simulation, obtaining packet dropping ratios and latency figures. This study allows the selection of optimal values for some network parameters, which will be necessary at the implementation stage. In particular, the obtained values of maximum latency will be used to select the appropriate values for the age-based packet dropping mechanism, implemented to avoid live-locks.

Our results lead to the conclusion that keeping in-transit packets waiting for too long for the allocation of output ports is counterproductive. This contention results in a backpressure that causes the dropping of packets at the injection queues. In most of the experiments, a maximum waiting time of one cycle leads to the best performance both in terms of the number of dropped packets and latency. The most aggressive policy, zero-waiting, is only slightly better for saturated networks, a situation that is not expected to happen in an actual SpiNNaker system with an over-dimensioned network.

We have performed also an evaluation of the system under fault scenarios. To do so, we have modelled the network taking into account broken (bi-directional) links. The SpiNNaker network performs acceptably well with a 0.5% of the links failing. When this value grows to 2% of the links, the system experiences severe performance degradation. However, note that our simulated model is unaware of the network failures, while the actual SpiNNaker system will be aware of them and will route the packets through *trusted* paths. Thus, a lower degree of degradation of the system is expected when using the actual system.

As future work we expect to perform more complex evaluations of the system with different failure and traffic models. Moreover some other different mechanism and topologies will be evaluated in order to improve the behaviour of the system when running in correct scenarios as well as when there are failures in the network.

ACKNOWLEDGEMENTS

This work has been supported by the Ministry of Education and Science (Spain), grant TIN2007-68023-C02-02, and by grant IT-242-07 from the Basque Government. The Spinnaker project is supported by the UK Engineering and Physical Sciences Research Council, partly through the Advanced Processor Technologies Portfolio Partnership at the University of Manchester, and also by ARM and Silistix. Steve Furber holds a Royal Society-Wolfson Research Merit Award. Javier Navaridas is supported by a doctoral grant of the UPV/EHU.

REFERENCES

[1] P Dayan and L Abbott, Theoretical Neuroscience. Cambridge: MIT Press, 2001.

[2] T Elliott and N Shadbolt, “Developmental robotics: Manifesto and application,” Philosophical Trans. Royal Soc., vol. A, no. 361, 2003.

[3] S Furber, S Temple, and A Brown, “On-chip and inter-chip networks for modelling large-scale neural systems,” in Proc. International Symposium on Circuits and Systems, ISCAS-2006, Kos, Greece, May 2006.

[4] S Furber, S Temple, “Neural Systems Engineering”. Journal of The Royal Society Interface 4(13), pp 193-206, April 2007

[5] MM Khan et al. "SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor". Proc. 2008 International Joint Conference on Neural Networks (IJCNN2008).

[6] LA Plana et al. “A GALS Infrastructure for a Massively Parallel Multiprocessor”. IEEE Design & Test of Computers, Volume: 24 , Issue: 5, pp. 454 - 463, Sept.-Oct. 2007

[7] LA Plana et al. “An on-chip and inter-chip communications network for the spinnaker massively-parallel neural net simulator,” in Proc. Second ACM/IEEE International Symposium on Networks-on-Chip (NoCS 2008), 2008, pp. 215 – 216.

[8] A Rast et al. “Virtual synaptic interconnect using an asynchronous network-on-chip,” in Proc. 2008 Int’l Joint Conf. on Neural Networks (IJCNN2008), 2008.

[9] FJ Ridruejo, J Miguel-Alonso. “INSEE: an Interconnection Network Simulation and Evaluation Environment”. Lecture Notes in Computer Science, Volume 3648 / 2005 (Proc. Euro-Par 2005).