# SWORDFISH: A SIMULATOR FOR HIGH-PERFORMANCE NETWORKS

Mondrian Nüssle, Holger Fröning, Ulrich Brüning

Computer Architecture Group, University of Mannheim, Germany
D-68131 Mannheim, B6, 26
email: {nuessle,froening,bruening}@uni-mannheim.de

## Abstract

SWORDFISH (Simple Wormhole Routing and Fault Injection on Simulated Hardware) is a simulator to explore the design space of high-performance networks. The simulator features a high modularity and is configurable by plug-ins. The simulated network is scalable to a large number of nodes and its modules are parameterizable to model timing, delays and buffer sizes. A simple and universal method to generate communication patterns is based on the single program multiple data (SPMD) programming model. Various topologies from distributed to centralized switches with a extendable variety of routing and arbitration functions can be generated easily. Extensive possibilities to collect both performance and statistical data gives the network designer a large set of data to prove the design decisions. Accuracy is proven by comparing the simulation results to real performance measurements of an existing network.

**Keywords:** High Performance Computing and Networking, Modelling and Simulation, Interconnection Networks, Cluster Computing

## 1 INTRODUCTION

The SWORDFISH project was started to explore the design space for a next-generation interconnection network (IN) based on the work of a previous existing network, the ATOLL network ([1]). The simulation should be scalable to large networks. Flexibility to incorporate new modules, design decisions, routing algorithms, etc. is very important. A major goal was the possibility to explore decisions in the areas of topologies, routing functions, buffer sizes, buffer management, flow-control, switching/arbitration strategies and fault tolerance features. Also, insights into the consequences of design decisions regarding head-of-queue blocking and deadlock freedom/removal were of interest. The simulation accuracy should be proven by comparing the simulation results to the real performance data of the ATOLL network.

There have been many projects on the topic of simulation of networks and of wormhole-switched networks. Examples of simulators for wormhole networks are [2],[3], [4] and [5]. RubinLAB provides for a detailed view of the single switch but lacks flexibility and support for large scale simulations. Many simulations are targeted strictly at packet switched networks respectively the Internet or TCP/IP based networks ([6], [7], [8], [9]). In recent time adaptive routing algorithms and multicast algorithms have seen significant work ([5], [10]). Often only a limited number of topologies is supported (for example k-ary n-cubes, [10], [11]). Traditionally, workloads for network simulators have often been modeled as stochastic processes of varying complexity ([11]) or trace generated ([12]). Some simulators provide a library for developing complex traffic patterns instead of simulating only trivial built-in standard patterns (e.g. [13]). Others provide a library for the implementation of user-defined network protocols ([14], [9]). Execution-driven workloads have not been used very often in the context of interconnection simulation. None of the simulators available to us met all the criteria of our necessities.While some projects provide a lot of flexibility ([11]) those solutions are targeted at a different field of application. This state of the art analysis led to the decision to build a new modular and scalable interconnect simulator to answer our questions for the design of high-performance networks for parallel and cluster computing.

SWORDFISH addresses source-path routed System Area Networks (SANs) employing wormhole- or virtual-cut-through switching, including intermediate switching methods. The different switching schemes are facilitated through different buffer and arbiter modules. The component oriented architecture of SWORDFISH also supports other routing and switching strategies but they have not yet been implemented. A high speed cross-bar switching element is chosen as the core element of the simulated network. Any node-degree and topology is supported. While k-ary n-cubes are a very popular family of topologies that are well supported by SWORDFISH, other topologies like trees or even highly irregular topologies can be simulated. Dynamically changing the topology is supported as a means to model link and node faults.

Another feature unique to SWORDFISH is the execution-driven approach which leverages the popular MPI (message passing interface) standard ([15]). This approach

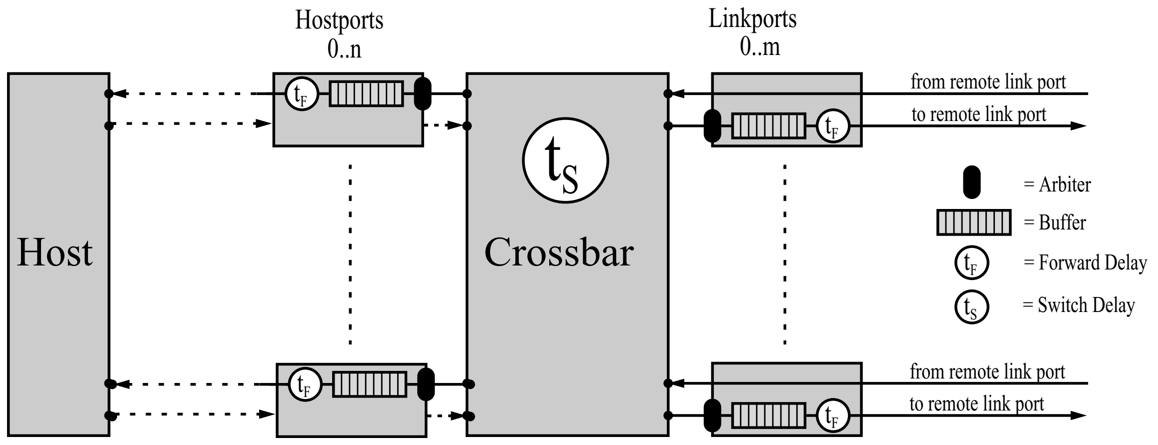


*Figure 1:SWORDFISH hardware model*

makes it easy to verify results from simulation runs against real networks (see Section 3). Also, users that are familiar with the SPMD paradigm of parallel programming and message passing find it easy to develop workload applets. In addition to the simulator itself, several tools and interfaces have been developed to be able to analyze the results of simulator runs as well as to provide for interactive simulation with a GUI. All input and output is organized as XML or HTML allowing integration with other tools. The included topology generation tool is especially useful when working with large scale networks.

Since qualified design decisions should be enabled for larger networks, special care was taken to leave the simulation on an abstraction level high enough for high-speed simulation while also taking care to program the simulator as an efficient C++ application. Flexibility is guaranteed by the component and plug-in based architecture of SWORDFISH. Together with the afore mentioned features, simulations of different scenarios can be achieved in a user-friendly way.

The remaining paper is organized as follows. Section 2 describes the SWORDFISH design. Section 3 presents first results from the SWORDFISH project. Finally a conclusion is drawn and an outlook presented.

## 2 SWORDFISH DESIGN

Figure 1 depicts the basic hardware model from which the SWORDFISH simulation model has been derived. Each node includes a crossbar as switching element. An arbitrary number of Linkports connect the node to other nodes of the interconnection network (IN). On the other side a number of ports connect the crossbar to the host. Each of these hostports (i.e. ports connecting the computing resources to the network) can be used by software to inject or extract messages to respectively from the network. Most of the functionality is based on plug-ins, which may be selected at runtime. The plug-in approach also enables plug-ins that model specific parts of a system in great detail, even down to a description of individual hardware components using SystemC ([16]).

The simulation core consists of objects for hostports and linkports, for the associated buffers and arbiters. The most important parts of these are the different buffer objects. There are six plug-in interfaces actually implemented, each for a different task. These are the SimulationController interface (for user interfaces in a general sense), the TrafficController interface (for traffic patterns), the Statistic interface (for statistical analysis), the Router interface (for routing algorithms), the LinkArbiter interface and the Arbiter interface (for arbitration algorithms).

All parametrization and input configuration for core functionality and plug-ins is done using XML files enabling editing as plain-text as well as more sophisticated methods involving XML tools. Some of the parameters are node degree, link throughput, switch delay, link delay, host-interface start-up delay and many more. Output can be generated as HTML report. Topologies are also described in XML files. While it is possible to write a topology file by hand, the included topology generation tool is much more convenient. The topology generation tool only supports regular topologies, but irregular topologies can for example be derived from a regular topology by removing links. Using a mapping file, it is further possible to specify which workload applet should be executed on which node. The individual parts of the system are described in the following paragraphs; first the network core components and then the rest of the system components.

**Simulation kernel and event processing.** Events are stored in the central event queue. The *simulate* function removes events and calls the dispatch method of the event object. This method in turn calls the event processing function either of a buffer object or a different plug-in (Figure 2). The processing function can generate new events that are passed back to the event queue. Once the operation is completed, registered notification functions are called. Such a function is for example useful to gather simulation output data every time a packet is delivered or any other event happened.

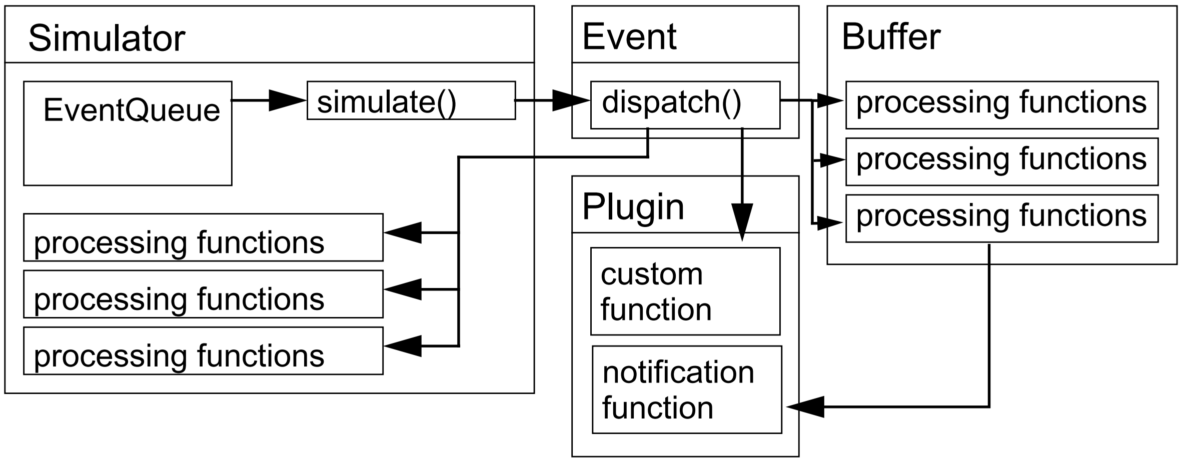


*Figure 2:Simulation kernel*

**Buffers.** Several buffer classes implement buffering at the point messages enter or leave the network, as well as on intermediate steps.
**Arbiter and Link Arbiter.** The arbiter plug-in interfaces for (main) arbiters and link arbiters share the same parent interface *Arbiter*. The arbiter is the core of the switching simulation as it selects which packet or parts of packets (flow control units, *flits*) may progress through the current node. The link arbiter can be used to multiplex several virtual channels on one link. In addition to the core functionality some utility functions are also available. There are two arbiter plug-ins for linkports and one main arbiter currently included. For the linkports these are FIFO and Round-Robin, for the main arbiter only FIFO is available.
**Hostport and Linkport.** A hostport consists of a number of buffers for injection and ejection. If those buffers ever get full, blocking occurs. Linkports are a little more complicated and include a link arbiter in addition to the buffers. Figure 3 depicts the logical relationship of the different buffers, ports and arbiters in one node of a network. These components make up the core of the network simulation.
**Router.** The router interface provides for the inclusion of routing algorithms. In order to give the plug-in the chance to create its routing table at the very beginning, the interface enables the routing plug-in to calculate its routing table before simulation commences. With every change of the topology (notably the removal and addition of nodes or links at runtime) an update of the routing is requested. The router plug-in must also provide the functionality to check for a valid route between two endpoints and provide the simulation core with a routing vector describing the source path route a particular message should take through the network. Current implementations of the arbiter only support source-path routing. In the future an arbiter supporting for example table based routing will enable different routing schemes for SWORDFISH. In this case the router component has to deliver the necessary information to the arbiter and the source path vector passed on to the TrafficController becomes an endpoint address to be interpreted by the arbiters.

Several different routing plug-ins are currently implemented. The first one, called *dijkstra*, implements the Shortest Path Algorithm by Dijkstra ([17]). As a side effect, the plug-in also covers dimension order routing if applied on regular meshes as produced by the topology generator script. Three other routing algorithms are first a slightly different implementation of the Dijkstra Shortest Path Algorithm, second a version of Up*/Down*-Routing([18]) and third West-First Routing ([19]). Finally there are different variants of deterministic routing algorithms that use virtual channels to provide deadlock freedom respectively to improve network behavior, for example the algorithm proposed by Dally and Seitz in their classic paper ([20]).
**Simulation Controller.** The simulation controller interface provides the possibility to develop interactive user interfaces for SWORDFISH. Such plug-ins are capable to suspend and resume the simulation loop at any time, also enabling breakpoints. A simulation controller may change global network state through the use of the core object hierarchy. This can be used for example to deactivate links or nodes in the midst of a running simulation, thus enabling the simulation of network faults. Currently, three controller plug-ins have been implemented. The dummy simulation controller module runs with no user interaction at all, but has the ability to terminate the simulation loop at a certain time (configurable through the XML scenario input file). Obviously this controller is most useful for batch like simulator runs. The console simulation controller extends the dummy controller with the ability to suspend the simulation loop using CTRL-C. Then the controller enters a console mode where the user can request some actions like setting a breakpoint or checking the current simulation time. The GUI plug-in is an example for a graphical user interface for SWORDFISH and is built using the Qt library (available from www.trolltech.no). The GUI is able to visualize 2-dimensional meshes and tori and to display the current state of the buffers and arbiters of the network. Other topologies are currently not supported by this plug-in, mainly due to difficulties designing an appropriate visualization of such a network. It is also

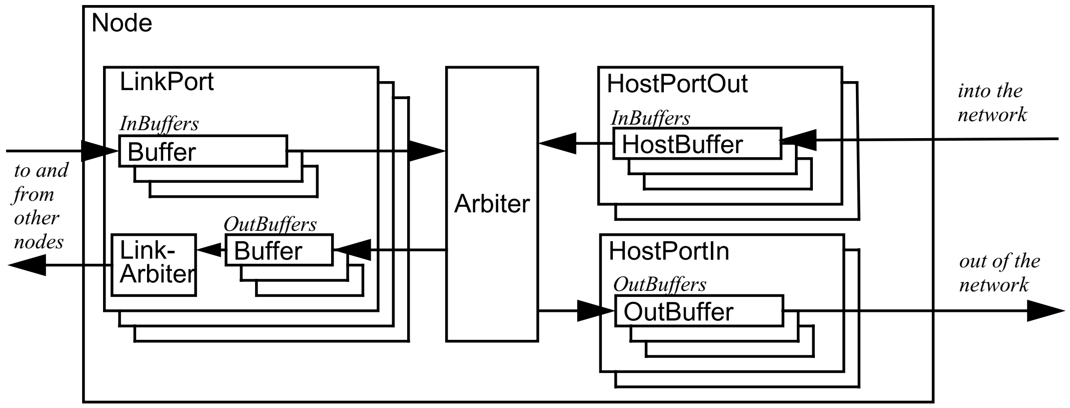
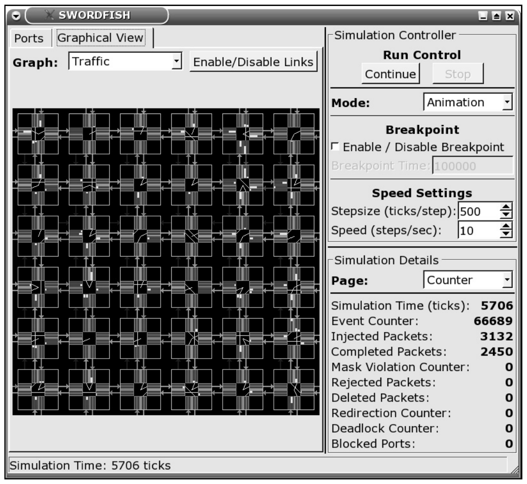


*Figure 3:Logical view of network core elements*

able to control the simulation speed and to run the simulation in single step mode. An additional feature is the possibility to disable and re-enable links in order to interactively simulate network faults. Figure 4 shows a screenshot of the GUI plugin.

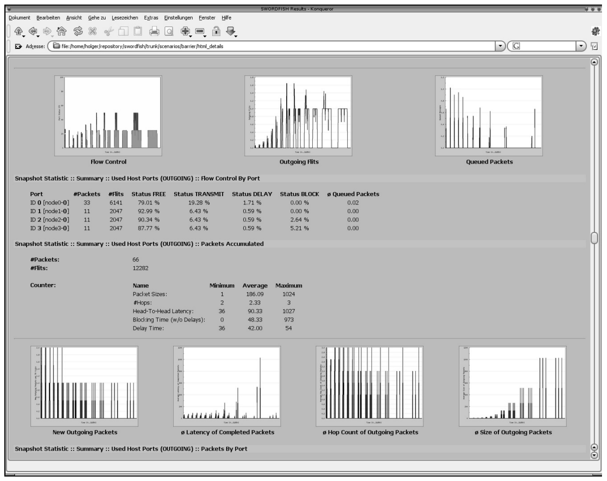


Simulation Time (ticks): 5706
Event Counter: 66689
Injected Packets: 3132
Completed Packets: 2450
Mask Violation Counter: 0
Rejected Packets: 0
Deleted Packets: 0
Redirection Counter: 0
Deadlock Counter: 0
Blocked Ports: 0

Simulation Time: 5706 ticks

*Figure 4: 2-D GUI plug-in*

**Statistic.** The statistic interface enables the generation of plug-ins collecting network runtime informations, evaluating them and finally providing the user with reports about the simulation run. Statistic plug-ins can collect very detailed data about every individual packet if they wish. Included is also the possibility to take snapshots of the network state which can be useful for a plug-in, for example a graphical user interface for SWORDFISH, which is implementing both the statistical interface as well as the simulation controller interface. All gathering of information is based on the ability of the plug-in to register itself for events. At the end of the simulation run the plugin can process and write-back the collected data. Currently two statistic plug-ins are available. The basic statistic plug-in writes general output to the console. The HTML statistic plug-in collects detailed information about every port and connection of the network and shows the development of some statistical figures over the simulation time. For performing this task the plug-in takes snapshots of the network state and stores them until the end of simulation (Figure 5).


Flow Control
Outgoing Flits
Queued Packets
New Outgoing Packets
ø Latency of Completed Packets
ø Hop Count of Outgoing Packets
ø Size of Outgoing Packets

Snapshot Statistic :: Summary :: Used Host Ports (OUTGOING) :: Flow Control By Port

| Port | #Packets | #Flits | Status FREE | Status TRANSMIT | Status DELAY | Status BLOCK | ø Queued Packets |
|---|---|---|---|---|---|---|---|
| ID **0** [node0-**0**] | 33 | 6141 | 79.01 % | 19.28 % | 1.71 % | 0.00 % | 0.02 |
| ID **1** [node1-**0**] | 11 | 2047 | 92.99 % | 6.43 % | 0.59 % | 0.00 % | 0.00 |
| ID **2** [node2-**0**] | 11 | 2047 | 90.34 % | 6.43 % | 0.59 % | 2.64 % | 0.00 |
| ID **3** [node3-**0**] | 11 | 2047 | 87.77 % | 6.43 % | 0.59 % | 5.21 % | 0.00 |

Snapshot Statistic :: Summary :: Used Host Ports (OUTGOING) :: Packets Accumulated

**#Packets:** 66
**#Flits:** 12282

**Counter:**

| Name | Minimum | Average | Maximum |
|---|---|---|---|
| Packet Sizes: | 1 | 186.09 | 1024 |
| #Hops: | 2 | 2.33 | 3 |
| Head-To-Head Latency: | 36 | 90.33 | 1027 |
| Blocking Time (w/o Delays): | 0 | 48.33 | 973 |
| Delay Time: | 36 | 42.00 | 54 |

Snapshot Statistic :: Summary :: Used Host Ports (OUTGOING) :: Packets By Port

*Figure 5: HTML Statistics*

**Traffic Controller.** The traffic controller interface is responsible for the complete user part of the communication model, i.e. the injection of packets into the network hardware and the reception of the packets from the hardware. The interface contains only a few methods, but these methods are powerful enough to simulate any traffic pattern SWORDFISH comes with one traffic controller plug-in, implementing an MPI-like API for user programs. The user writes a normal C(++) program using the defined API which becomes a simulation applet. The user code is then compiled into a shared library object using the standard C Compiler. The traffic controller plug-in opens such an object file upon start of the simulation and then spawns a thread for each virtual process taking part in the simulation. This module was designed to spawn and handle very large amounts of threads and therefore many simulated processes. In test scenarios under Linux the maximum amount of

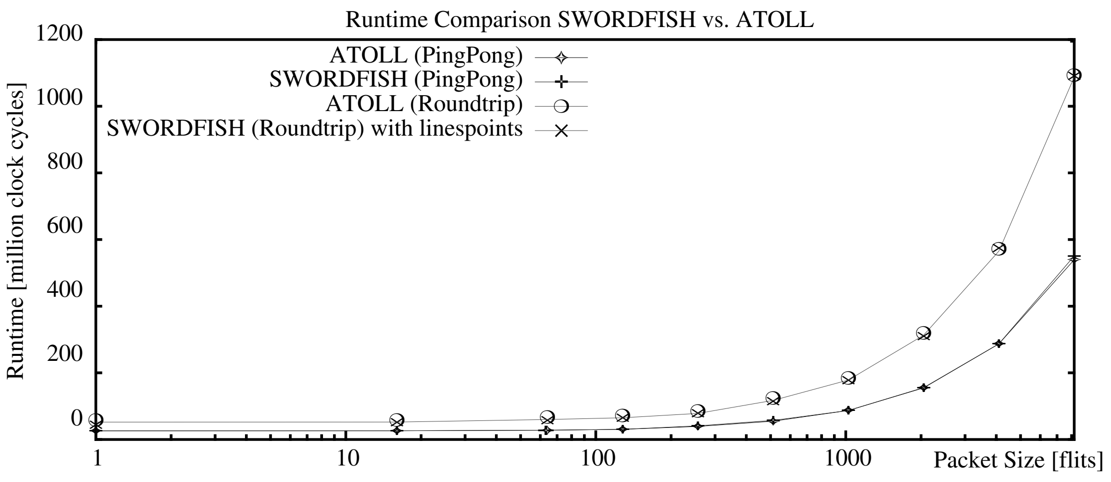


*Figure 6:ATOLL vs. SWORDFISH timing results*

spawned threads was more than 50000 (Linux 2.6 and Native Posix Threads Library, NPTL [21]).

A big advantage of the API and the approach of writing normal programs with a *main()* function for defining traffic patterns is the relative ease to execute the programs on real networks using MPI like libraries. The only adjustment is to map the API functions to functions available for the physical network. Conversion of programs to use MPI is very straight forward and it is equally simple to port programs to other message passing APIs. The functions included into the API are summarized in Table 1. Readers familiar with MPI-1 message-passing will recognize the close relationship to real message passing libraries (though only a subset of functions is available in comparison with a full-blown MPI-1 implementations). If desired, it is easy to extend this interface to support more communication functions, like for example collective operations.

| Function name | Description |
|---|---|
| Init/Finish | Initialize and finalize system |
| Size/Rank | Return number of processors/return number of own processor |
| Time/ConsumeTime | Return current simulation time/model computational time |
| Send | Send a message/packet |
| Recv/RecvFrom/RecvTag | Receive a message/packet |
| Probe/ProbeFrom/ProbeTag | Check for new message(s) |

*Table 1: SWORDFISH workload API*

**Topology generation.** A topology generation script called `create_topology.py` has been developed, creating XML topology files as used in the scenario files. The generator is implemented in Python([22]) and supports regular meshes, tori and hypercubes. It writes the output XML file to the console. Currently, other topologies have to be entered by hand or derived from regular topologies generated with the generator.

## 3 FIRST RESULTS

In a first evaluation of the simulator, the results that were forecast by simulation, have been verified with a real network, in this instance the ATOLL network.

First, buffer sizes of the simulator configuration were chosen to match the buffers of ATOLL. Next, the timing parameters had to be specified. The switching delay $t_s$ and the forwarding delay $t_f$ were taken from the ATOLL reference documentation. The offset and variable part of the time for message injection and removal were taken from measurements on an actual system featuring ATOLL ([23]). The measurement was performed using the Time Stamp Counter of the processor and converted into IN clock ticks (the time measurement unit of SWORDFISH). Then two MPI programs were written. One performed a simple ping-pong on two nodes and the other a ring round-trip on four nodes. This program was ported to the SWORDFISH API. Then these programs were run both on the real network and on the simulator. All programs performed 1000 iterations with packet sizes ranging from 1 to 8192 flits. On the real system the measurement was repeated 10 times and the minimum time was chosen for the analysis to account for preemption, interrupts and other causes of timing errors. The result of these tests are shown in Figure 6.

The simulation results show a very close accordance with the real network. Overall, they did not deviate more than 3% from the measured values. Unfortunately, no tests with larger configurations were possible because of the lack of a larger network.

Currently SWORDFISH is actively used to explore the design space of a new IN. Parameters that have been inspected include buffer sizes, flow control implementation, routing algorithms and impact of virtual channels.

## 4 CONCLUSION AND OUTLOOK

SWORDFISH makes up a perfect tool for the design space exploration of INs. The different features of SWORDFISH that have been presented in this paper are specifically designed for this purpose. Execution-driven simulation enables sophisticated workload modelling. The plug-in system allows for flexible simulation and using XML as a standard format makes it easy to develop scripts (e.g. in Python) to control and manage a large amount of simulation scenarios. SWORDFISH has already been used to explore the design space of interconnection networks and will provide the means for further insights into this area of interest. Very large configurations of ATOLL and improved networks have been simulated. The possibility to include SystemC models of hardware modules enables a mixed abstraction layer simulation and verification of these modules in the context of a larger system. This has already been used to integrate a hardware based barrier network into the SAN. Though this has proven to be useful, more work has to be spent to further advance this method mostly in terms of better interfacing as well as performance optimizations.

While SWORDFISH has been primarily developed to analyze source-path routed networks, the next steps will broaden this to handle non source-path routing. Also, the support for networks with a more centralized switching resource will be significantly improved. Once these improvements are done, analysis and comparisons of a range of current high-performance networks as used in cluster computing becomes possible and will be interesting to conduct. Again, emphasize will be on large configurations.

## 5 REFERENCES

[1] U. Brüning, H. Fröning, P. R. Schulz, L. Rzymianowicz, ATOLL: Performance and Cost Optimization of a SAN Interconnect, *IASTED Conference: Parallel and Distributed Computing and Systems (PDCS)*, Cambridge, USA, 2002

[2] G. Lustig, W. Obelöer, A.C. Döring, RUBIN-Lab: Eine Simulationsumgebung für regelbasierte Routing-Algorithmen, *Tagungsband der Fachtagung "Architektur von Rechensystemen - ARCS'99"*, Jena, Germany, 1999, 210-219

[3] R. Bogrodia, Y. Chen, M. Gerla, et al., Parallel Simulation of a High-Speed Wormhole Routing Network, *Proceedings. 10th Workshop on Parallel and Distributed Simulations*, Philadelphia, PA,1996, 47-56

[4] R.V. Boppana, S. Chalasani, and C.S. Raghavendra, Resource Deadlocks and Performance of Wormhole Multicast Routing Algorithms, *IEEE Transactions on Parallel and Distributed Systems, 9*(6), 1998.

[5] L. Schwiebert, A Performance Evaluation of Fully Adaptive Wormhole Routing including Selection Function Choice, *IEEE International Performance, Computing, and Communications Conference*, Phoenix, AZ, 2000, 117-123

[6] S. Keshav, REAL - A Network Simulator, *Technical report*, University of California, Berkeley, 1988

[7] S. McCanne, S. Floyd, ns-Network Simulator, http://www.isi.edu/nsnam/ns/, 2002

[8] L. S. Brakmo, L. L. Peterson, Experiences with Network Simulation, *SIGMETRIXC 96*, Philadelphia, PA, 1996

[9] A. Terzis, K. Nikoloudakis, L. Wan, L. Zhang, IRL-Sim: A General Purpose Packet Level Network Simulator, *Proc. of 33rd Annual Simulation Symposium*, Washington D.C., 2000

[10] The Chaos Router Simulator, http://www.cs.washington.edu/research/projects/lis/chaos/www/simulator.html

[11] J. Rexford, W.C. Feng, James Dolter, K. G. Shin, PP-MESS-SIM: A flexible and Extensible Simulator for Evaluating Multicomputer Networks, *IEEE Transactions and Parallel and Distributed Systems*, *8*(1), 1997

[12] J.M. Garcia, J.L. Sanchez, P. Gonzalez, Pepe: A trace-driven simulator to evaluate reconfigurable multicomputer architectures. Springer Lecture Notes in Computer Science 1184, 1996, 302-311

[13] D. F. Bacon et al., NEST: A Network Simulation and Prototyping Tool, IBM, T.J. Watson Research Center, 1988

[14] M. Allman, A. Caldwell, S. Ostermann, ONE: The Ohio Network Simulator, *Technical Report TR-19972*, Ohio University, Athens, OH, 1997

[15] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, www.mpi-forum.org, 1994

[16] SystemC Language Reference Manual, Version 2.1, www.systemc.org, 2005

[17] E. W. Dijkstra, A Note on Two Problems in Connection with Graphs, *Numerical Mathematics 1*, 1957, 269-271

[18] M. D. Schroeder et al., Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links, *Technical Report 59*, System Research Center of Digital Equipment Corporation,1990

[19] C. J. Glass, L. M. Ni, The turn model for adaptive routing, *Proceedings of the 19th Annual International Symposium on Computer Architecture*, 1992

[20] W. J. Dally, C. L. Seitz, Deadlock-free message routing in multiprocessor interconnection networks, *IEEE Transactions on Computers, 36*(5), 1987, 547-553

[21] U. Drepper, I. Molnar, The Native POSIX Thread Library for Linux, people.redhat.com/drepper/nptl-design.pdf, 2005

[22] Python Programming Language 2.4, http://www.python.org/, 2005

[23] H. Fröning, M. Nüssle, D. Slogsnat, P. R. Haspel, U. Brüning, Performance Evaluation of the ATOLL Interconnect, *IASTED Conference: Parallel and Distributed Computing and Networks (PDCN)*, Innsbruck, Austria, 2005