Implementation of an MPI Interface for SWORDFISH

Christian Leber

Computer Architecture Group
Institute for Computer Engineering
University of Mannheim

Table of Contents

1 Introduction

1.1 Motivation

This work was started, because with swordfish there is an existing simulator for the simulation of ATOLL/EXTOLL, but it's traffic generators are not sophisticated enough to cause traffic patters that are close enough to real world applications.

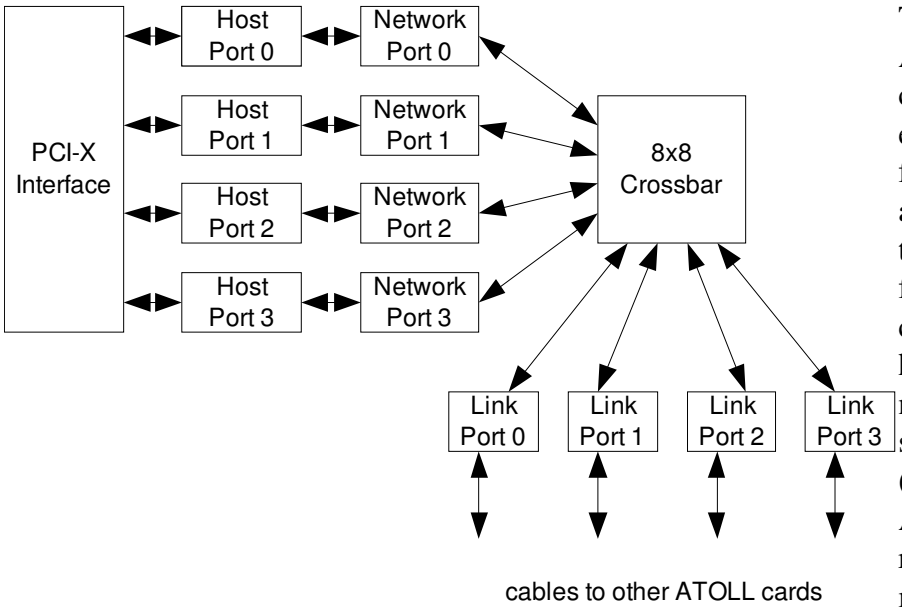
Therefore the idea was to develop a binding between SWORDFISH and existing software for clusters. The best way to accomplish this is to add MPI support to SWORDFISH, so that it's possible to run any software that makes use of MPI.

1.2 ATOLL

ATOLL [ATOLL] is a SAN[1] designed and developed by the Computer Architecture Group as a fast Cluster interconnect for cost efficient “off the shelf” clusters.
To name a few properties of ATOLL:

- ATOLL is designed for low latency, because most messages in clusters are short.
- Of course high throughput was also important, the link bandwidth is 250 MB/s
- The ATOLL chip itself is a “network on a chip” because each chip is a network interface for the host system and a network switch at the same time. The ATOLL chip is the only complex or unique component that is required in an ATOLL network, therefore it is very cost efficient.

The throughput and latency of ATOLL was state-of-the-art when the first hardware got available.



The main component of ATOLL is a self-routing 8x8 crossbar switch, therefore 8 elements can be connected full-duplex to it. Four of these are link ports for connections to other ATOLL cards and four are connected to so called host ports. The four host ports act as independent network devices for the software, so on each node (when each node has 1 ATOLL card) it's possible to run four processes that can make use of ATOLL. In 2002 this was no disadvantage, because no systems with more than four CPUs were in sight, that were cheap enough to be useful for a cluster. Today the situation would be different in this respect. The advantage of this solution is that

1 System Area Network

this way the hardware can be mapped into the userspace and therefore userspace/kernelspace context switches can be avoided.
The four Link Ports are making it possible to connect the nodes of an ATOLL cluster in a 2D torus, that is the optimal configuration that can be accomplished with four links per node.

EXTOLL is its designated successor and at the moment under active development.

1.3 SWORDFISH

Is a network simulator that was started as part of the diploma thesis [SWF_HS] of Holger Sattel, the abbreviation stands for: Simple Wormhole Routing and Fault Injection on Simulated Hardware

SWORDFISH supports the simulation of large networks with thousands of nodes.
The design is based on a plugin architecture that makes it very flexible to research different simulated network components.
The original version of SWORDFISH simulated accurately a ATOLL network, later versions were extended with the help of said plugin system to also simulate EXTOLL.

SWORDFISH is a event based simulator, every event in the simulator is added to a priority queue and processed in order of the time when said event should happen.

The configuration is flexible, because nearly everything is configured in .xml files that are read by swordfish on startup. Furthermore it also has a GUI that visualizes the data flow and is able to produce HTML reports.

1.3.1 The problem with SWORDFISH

The mechanism to get input for simulations is the so called trafficcontroller that starts as many threads as there are nodes that should be simulated in the network. This threads can make use of a few calls that mimic a very simple MPI interface. This approach has the following problems:

- programs that should be simulated have to be written specifically for this interface.
  (it's certainly possible to map them to a very simple MPI API and run very simple MPI applications)
- they all run as threads inside the simulator, this causes additional problem for less than simple applications
- writing test programs that causes traffic patterns that are indistinguishable from real MPI applications is a very hard problem and bound to fail.

For this reasons it is desirable to run real MPI applications with real MPI Implementations on top of the simulator, exactly that was the intension for this work.

Hence a new trafficcontroller had to be developed, with the following requirements:

- The nodes should connect with TCP/IP to the simulator, because this does also allow it to use more than one System for the simulations, this is especially useful if the MPI applications that should be used with the simulator need a lot of resources.
- the trafficcontroller has to keep track of the time

2 Problems and choices

2.1 Different MPI Implementations

It's important to point out that MPI doesn't name any specific software implementation, but it specifies an API, therefore there are different implementations and for this work an implementation had to be chosen.

To name a few:

- MPICH2 (http://www-unix.mcs.anl.gov/mpi/mpich2/)
- Scali MPI (http://www.scali.com)
- LA-MPI (http://public.lanl.gov/lampi/)
- Open MPI (http://www.openmpi.org)

Following features were required:

- sources freely available, because otherwise debugging is not possible
- the implementation should be in active development
- it should be a full (or at least close to) MPI-2 implementation

In the end there only MPICH2 and Open MPI met this requirements.

2.1.1 MPICH2

First MPICH2 was chosen, because prior work in form of a “ch3 channel” implementation for the ATOLL network was done by Sven Stork in the research group.
Such a “ch3 channel” driver has to be developed for every interconnection network that is used with MPICH2.

After a lot of debugging the idea was given up, because the implementation of a “ch3 device” driver for mpich2 is a tedious task and did not work reliable, there are many problems associated with this:

- undocumented macros
- preprocessor macros in multiple “layers” so that it's hard to find out what a macro actually does
- a lot of functionality has to be implemented in each “ch3 channel” driver before it's possible to get anything working

While trying to implement this in MPICH2 it got also clear that this is an old code base with many other problems. The single problems on their own are no clear argument against using MPICH2, but in their sum they are an indication that MPICH2 should not be used for future work.

2.1.2 Open MPI

Open MPI[HET_MPI] was started in 2004 with a completely new code base, but developed by the

developers of the following existing MPI implementations and their design ideas, namely:

- FT-MPI (University of Tennessee)
- LA-MPI (Los Alamos)
- LAM/MPI (Indiana University)
- PACX-MPI (HLRS, University of Stuttgart)

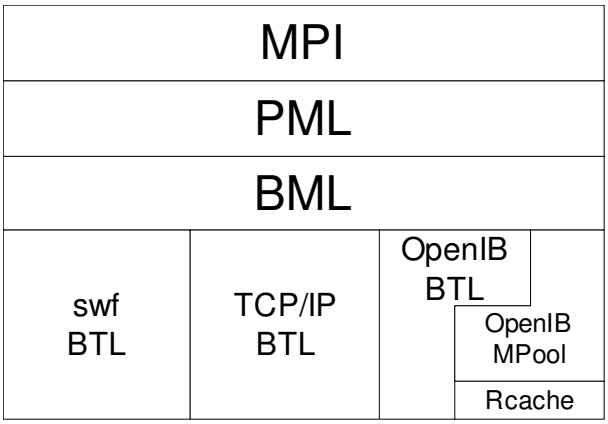
Therefore Open MPI has a clean code base and the usage of it with modern interconnection networks was planned from the beginning, without any legacy.
(the named older MPI implementations are not developed anymore, but they get bug fixes, because there are existing user bases)

A few more properties of Open MPI before explaining some parts of it's architecture:

- Open MPI is licensed under a Open Source license, that is BSD style
- Open MPI 1.0 was released in march 2006
- as November 14, 2006 Open MPI powers the #6 in the Top500 list of supercomputers
- Open MPI is based on an OOP design despite it's developed in ANSI C

2.1.3 Network support in Open MPI

Because of the component architecture of Open MPI, support for interconnection networks is loaded as modules and even multiple of this modules may be used at the same time to increase bandwidth even further, as illustrated here:



This is of course for illustration only, using SWORDFISH and Infiniband at the same time would defeat the purpose of a simulation.

The different layers and their functionality:

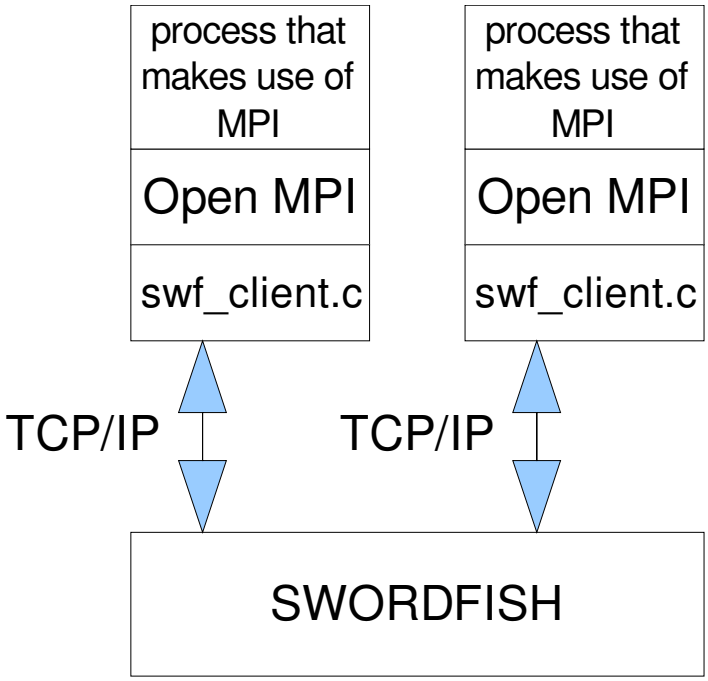
- **Point-to-point Management Layer – PML:** Implements MPI Point-to-point semantics on top of BTL
- **BTL-Management BML:** creates, discovers and manages the BTL modules
- **Byte-Transfer-Layer – BTL:** handles the actual data transfer from point to point

Therefore it was only necessary to develop a new BTL for SWORDFISH, because it can make use of

existing infrastructure. The important points for such an implementation are described later.

3 Overview

The whole infrastructure has the following outline:



The usage of a TCP/IP connection allows it to run the processes on different physical systems.

4 Implementation

4.1 The SockModel Protocol

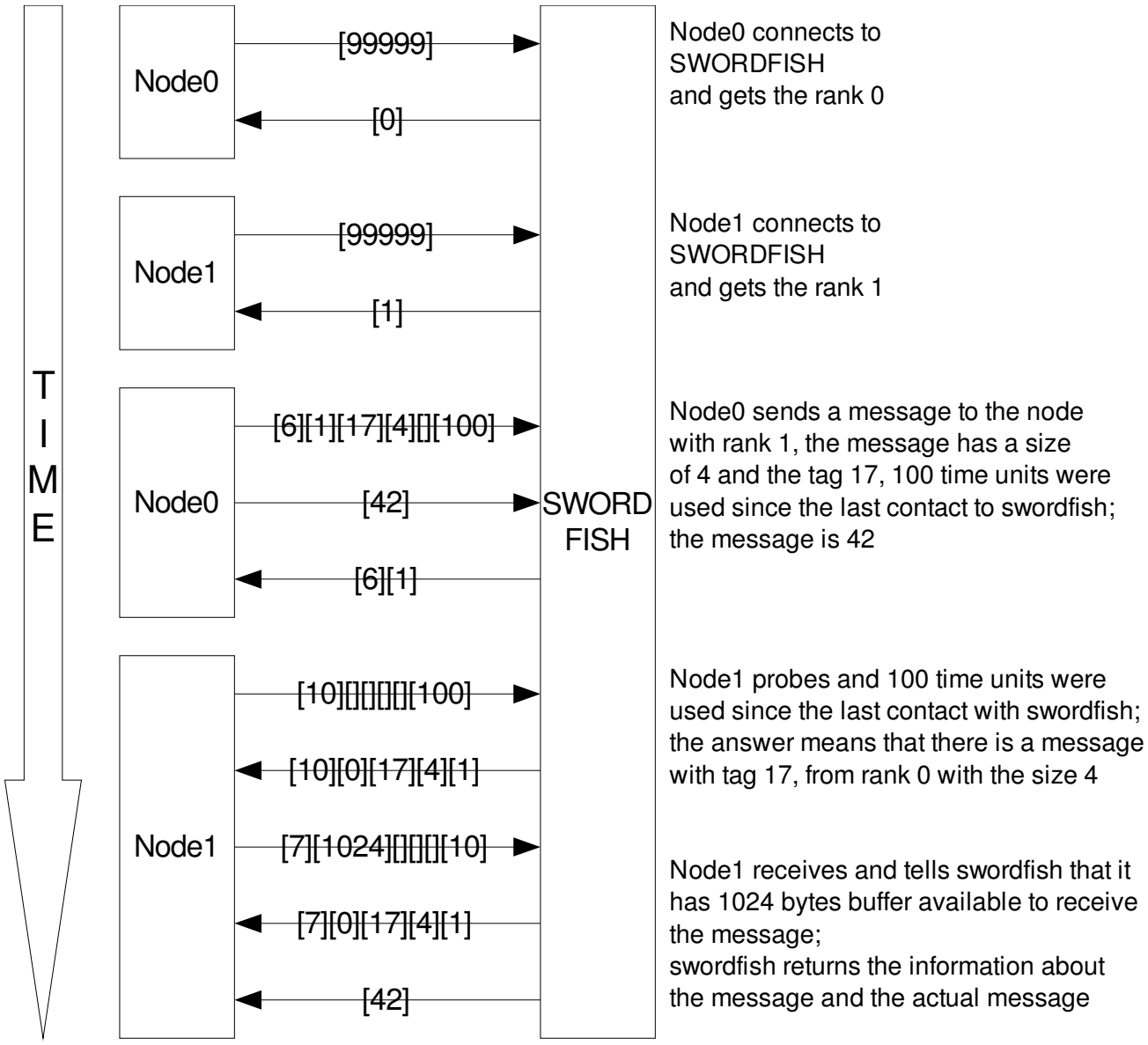
A relatively simple protocol was designed to ensure the communication between the simulator and the clients.
Basically the client can send commands to the simulator and it gets answers, all commands sent to the server consist of six unsigned 32 bit integers, all commands are initiated by the clients.
Each of this commands starts with the *request* that indicates the command, then 4 parameters are following (not all are used always) and at the end there is another 32 bit integer that contains the so called consumetime.
The consumetime is supposed to be the CPU time that was used since the last command was sent, the time unit is in cycles of the simulated hardware, for example 4ns.

When connecting each client has to send a rank as unsigned integer that it would like to acquire in the simulated network, but usually each client will send 99999 and therefore the rank will be set by the simulator. The simulator will answer with an unsigned integer that will contain the rank the client.

The requests:

- SWORDFISH_FINISH closes the connection
- SWORDFISH_SIZE returns the size of the network (number of nodes)
- SWORDFISH_RANK returns the own rank inside the network
- SWORDFISH_CONSUMETIME the nodes consumes the given amount of ticks
- SWORDFISH_SEND sends data to a given node, the data has to follow on the socket after the command was send
- SWORDFISH_RECV receives a message blocking
- SWORDFISH_RECVFROM receives a message blocking, but only from a given node
- SWORDFISH_RECVTAG receives a message blocking, but only when it has a given tag
- SWORDFISH_PROBE probes if there is a message available for the node
- SWORDFISH_PROBEFROM probes if there is a message available for the node from a given other node
- SWORDFISH_PROBETAG probes if there is a message available for the node with a given tag
- SWORDFISH_BARRIER_INIT initiates a barrier, the barrier id will be returned
- SWORDFISH_BARRIER_ENTER enters a barrier
- SWORDFISH_WAITING_START if there is a message waiting, then it will return the info about this message, so it will work exactly like SWORDFISH_PROBE. If there is no message ready, state of the node inside of SWORDFISH will go to the WAITS_FOR_MESSAGE state, this way SWORDFISH will send a message to the client when there is a new message; This functionality was added to avoid too much resources consuming probing.
- SWORDFISH_WAITING_STOP will end the WAITS_FOR_MESSAGE state

Example for such a communication:



The number in [number] describes a 32 bit unsigned integer that is sent,
when the brackets are empty the value isn't used

4.2 The Implementation of the SockModel in SWORDFISH

As mentioned SWORDFISH already had multiple so called trafficcontrollers, the problem is that they are multi threaded in an unusual way.
Threads used by SWORDFISH:

- 1 simulationscontroller thread
- 1 simulation thread
- N (depending on the number of simulated nodes) node threads

The simulationscontroller thread, that runs the GUI or a similar controller, is independent, but only one of the other threads can run at a time.

For the new SockModel trafficcontroller it was not possible to do it the same way, therefore three threads were used in the first place.

- 1 simulationcontroller thread
- 1 simulation thread
- 1 SockModel thread

That was the first big problem, because it was not possible to run only either the simulation thread or the SockModel thread, the problem was that none of the data structures in SWORDFISH were thread safe.

Therefore several data structures and functions had to be extended by mutexes and condition variables, while keeping everything compatible with the other trafficcontrollers.

First the SockModel trafficcontroller was developed like the other trafficcontrollers as plugin, but eventually this was a big problem, because much more interaction between the simulator thread and the SockModel thread got necessary, but the interface didn't provide this capabilities.
Hence the SockModel was moved to the simulation thread, instead of adding more overhead and complexity by adding this interfaces.
This is not a performance problem, because the SockModel and the simulation thread blocked each other relatively very often, but therefore all the measurements taken to make the simulation thread-safe could be removed again.

4.2.1 Keeping track of time

A central point of every simulation is keeping track of the time, because the required time shows how good the simulator is working with different possible parameters or configurations.
Therefore there is a time counter for every simulated node in the network and only events that are in the past of all nodes are processed.
The time proceeds either by network activity that adds specific amounts of time, for example 100 cycles for probing for a message or in the form of SWORDFISH_CONSUMETIME when computing time is used by a simulated MPI process.

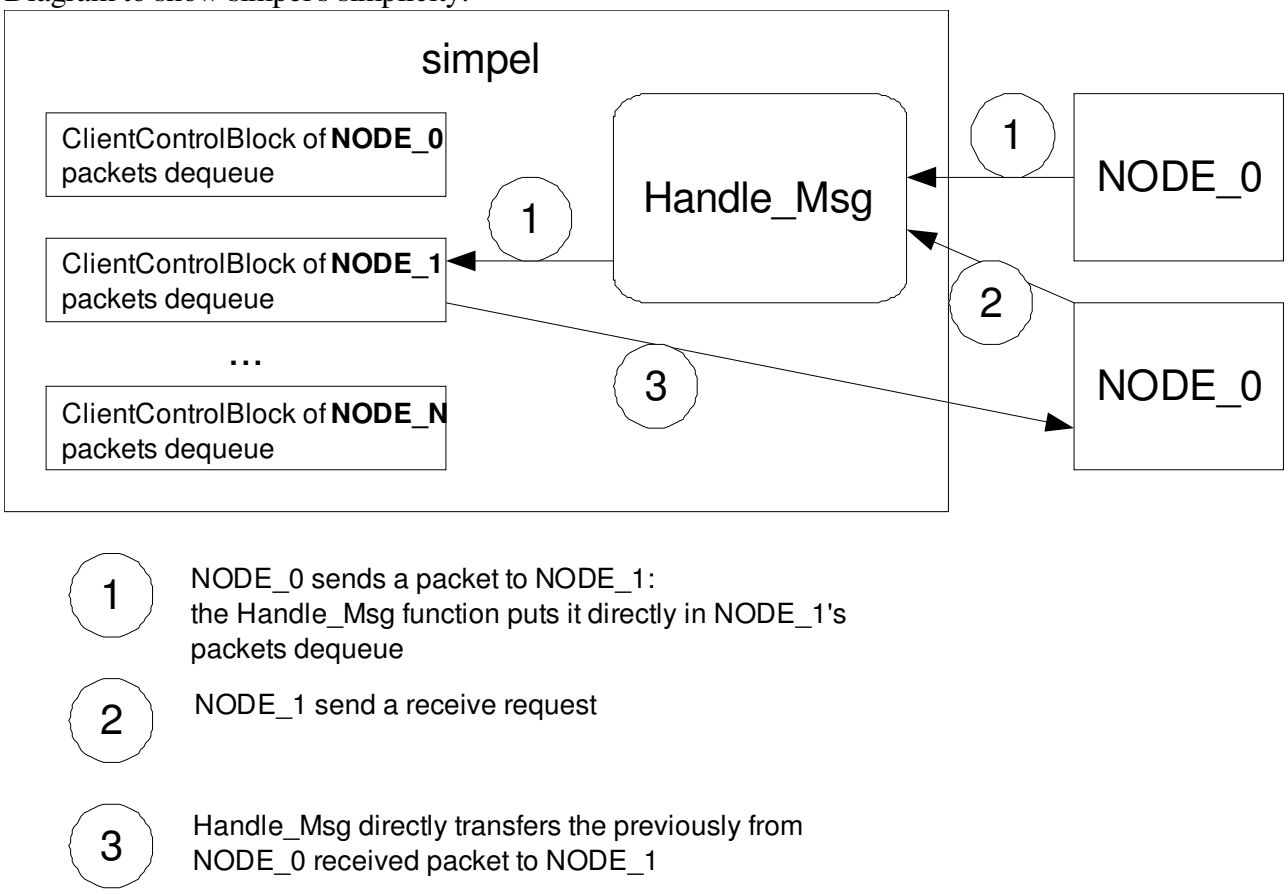
4.3 simpel – a simulator simulator

The idea of simpel is a “simulator-simulator”, it behaves exactly like SWORDFISH when the SockModel is used, but it doesn't really simulate a network. And that is exactly what it was written for, because while developing the different parts and there was a problem, it was unclear were it is.
The features are:

- relatively “bug free” because it's small and simple
- implements the SockModel Protocol as describes before
- shows a few stats on exit
- nothing more

A very important “feature” of simpel is that is simulates no time, so no deadlocks are possible because the time isn't advanced correctly, that is very handy when the reasons for a deadlock are unclear.

Diagram to show simpel's simplicity:



1 NODE_0 sends a packet to NODE_1:
the Handle_Msg function puts it directly in NODE_1's packets dequeue

2 NODE_1 send a receive request

3 Handle_Msg directly transfers the previously from NODE_0 received packet to NODE_1

4.4 The Implementation of the Open MPI BTL (bit transfer layer)

Open MPI provides the possibility to develop modules for it out of tree, this makes this task a little bit easier, because such a module will work with different versions of Open MPI, thought API stability is of course no guaranteed.

Because of the infrastructure Open MPI provides a good environment only few functionality has to be implemented to reach the point where the module works.

The only task that has to be accomplishes by each BTL is to transfer data from endpoint to endpoint without taking care about any management or similar things, because that is handled by a higher level.

4.4.1 Initialization

A BTL is a PML component and the initialization is done in:

```
mca_btl_base_module_t** mca_btl_swf_component_init(int *num_btl_modules,
                                                   bool enable_progress_threads,
                                                   bool enable_mpi_threads)
```

This function has to malloc memory for the required data structure:

```
mca_btl_swf_module_t**m =0;

*num_btl_modules=1;
m=malloc((*num_btl_modules)*sizeof(mca_btl_swf_module_t*));
m[0]=(mca_btl_swf_module_t *)&mca_btl_swf_module;
```

Also the connection to SWORDFISH is initiated in this function:

```
swfcom_init_struct(&(m[0]->con));
swfcom_connect_with_rank(&(m[0]->con),99999);
swfcom_get_rank(&(m[0]->con));
swfcom_get_size(&(m[0]->con));
```

On the more interesting side, a functionality of the PML is used her to tell the other processes about this endpoint, this does nothing more than make the SWORDFISH rank of this connection available to the other Open MPI processes.

```
mca_pml_base_modex_send(&mca_btl_swf_component.super.btl_version,
                  &(m[0]->con.rank), sizeof(uint32_t));
```

The communication for BTL is endpoint based, that means that the higher level (BML) will call the send function with the endpoint as parameter, the data structures for this have to be initialized in the following function:

```
int mca_btl_swf_add_procs(
    struct mca_btl_base_module_t* btl,
    size_t nprocs,
    struct ompi_proc_t **ompi_procs,
    struct mca_btl_base_endpoint_t** peers,
    ompi_bitmap_t* reachable)
```

For the setup of the endpoint only the rank in SWORDFISH is required, that is exactly the rank that was sent with `mca_pml_base_modex_send`.

```
for (i=0;i<nprocs;i++) {
      struct ompi_proc_t* ompi_proc = ompi_procs[i];
      mca_btl_base_endpoint_t* swf_endpoint;

      // do nothing for itself
      if(ompi_proc == ompi_proc_local())
            continue;

      uint32_t *peer_node_id;
      rc= mca_pml_base_modex_recv(&mca_btl_swf_component.super.btl_version,
            ompi_procs[i], (void*)&peer_node_id,&size);

      /* -> set reachable bitmap bit */
      ompi_bitmap_set_bit(reachable, i);
      swf_endpoint = OBJ_NEW(mca_btl_swf_endpoint_t);
      if(NULL == swf_endpoint) {
            return OMPI_ERR_OUT_OF_RESOURCE;
      }
      swf_endpoint->con=&(swf_btl->con);
      swf_endpoint->node_id=*peer_node_id;
      peers[i] = swf_endpoint;
}
```

This code sets for all nprocs nodes, besides itself a data structure from the type mca_btl_swf_endpoint_t up.

4.4.2 Sending a Message

This part is surprisingly simple, the function is defined as:

```
int mca_btl_swf_send(
    struct mca_btl_base_module_t* btl,
    struct mca_btl_base_endpoint_t* endpoint,
    struct mca_btl_base_descriptor_t* descriptor,
    mca_btl_base_tag_t tag)
```

The endpoint contains all the data we need to send the data in the descriptor:

```
swfcom_send(endpoint->con,endpoint->node_id,tag32,
    frag->segment.seg_len,(void *)(frag->segment.seg_addr.pval))
```

while swfcom_send it defined as:

```
unsigned swfcom_send(swf_conn *con, unsigned dest,unsigned tag,
    unsigned size, void* buffer);
```

4.4.3 Receiving a Message

It is necessary to probe for new message though Open MPI provides a framework for this, because the time in the simulator has to proceed, otherwise it may lead to a deadlock situation. This is one of the key differences to real networks, because in a real network the time proceeds anyway, but with a simulation this doesn't work.

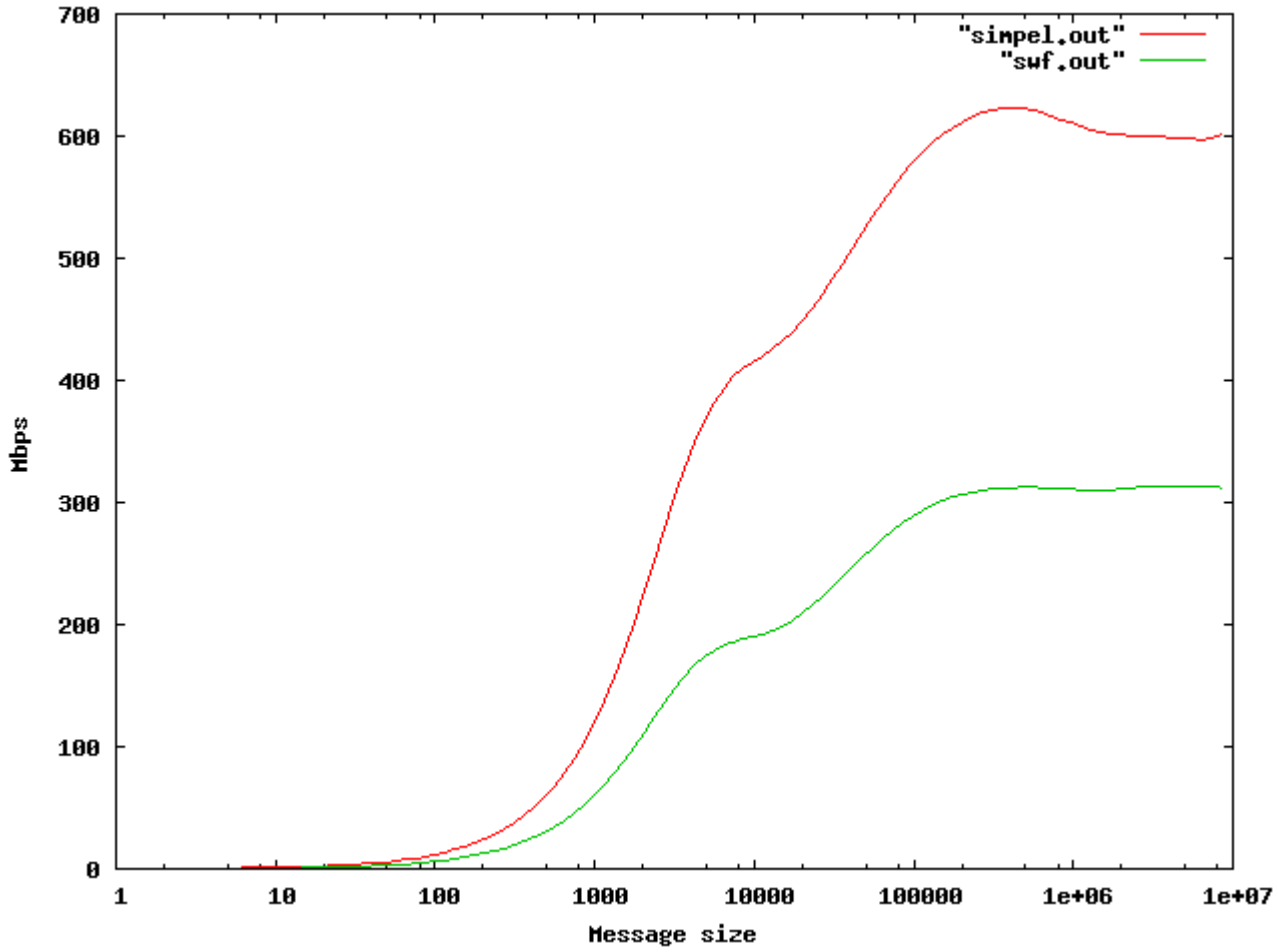
Hence in "***int mca_btl_swf_component_progress(void)***" probe requests are sent to SWORDFISH, because this will advance the time, but it's a problem to probe too often, because this may use too much resources, therefore the following mixed approach was taken:

- the first 3 times the progress function is called a probe request will be made
- after this a SWORDFISH_WAITING_START request will be made, that means that SWORDFISH will message the Open MPI process when there is a new message, then the progress function waits with epoll_wait for this for some time time that increases with every failed attempt. Actually it will wait per failed attempt 1 ms, but not more than 100ms. When epoll_wait returns a SWORDFISH_WAITING_STOP request is issued, to prevent unexpected data from SWORDFISH

5 Results

To verify the functionality of SWORDFISH with the new trafficcontroller multiple test programs were written, for example: roundtrip and all2all sending

Further Netpipe 3.6.2 was used as an example of an simple MPI program, it was run many times with different options. Netpipe is a really simple MPI program, it doesn't use more than send, receive and barrier. The following chart shows a run with the default Netpipe options and Open MPI with the SWORDFISH BTL module.



As you can see SWORDFISH (swf.out) is about half asfast as simpel (simpel.out).

Also the Intel MPI Benchmark was used, eventually not as benchmark, because the output is not easy to compare, but as test suite it makes use of much more MPI features than NetPipe.
Therefore the successful finishing of the Intel MPI Benchmark shows that MPI is fully working over SWORDFISH.

6 APPENDIX

6.1 Installation and usage of the Open MPI module

1. The Open MPI module is developed and tested with Open MPI 1.1, therefore it is suggested
2. In the Open MPI src dir:
   `./configure --prefix=<somedir> --with-devel-headers`
3. `make -j 4 && make install`
4. go to the scenarios/sockets/btl_swf dir:
   `sh admin/bootstrap`
   `./configure –prefix=<somedir>`
   `make && make install`
5. add:
   `export PATH=<somedir>:$PATH`
   to your ~/.bashrc
6. `source ~/.bashrc`
7. compile your mpi testprogram with:
   `mpicc testprogram.c`
   or compile NetPIPE with: `make mpi`
8. `echo localhost > hosts`
9. run `./simple 2`
   or `swf sockmodel_2.xml` in one window
10. `mpirun --mca btl swf -hostfile  hosts -np 2 <mpiapp>` in another window

7 Bibliography

[ATOLL]: U. Brüning H. Fröning P. R. Schulz L. Rzymianowicz, ATOLL: PERFORMANCE AND COST OPTIMIZATION OF A SAN INTERCONNECT, 2002

[SWF_HS]: Holger Sattel, A Scalable Generic Wormhole-Routing Simulator - SWORDFISH, 2004

[HET_MPI]: Richard L. Graham, Galen M. Shipman, Brian W. Barrett, Ralph H. Castain, George Bosilca, Andrew Lumsdaine, Open MPI: A High-Performance, Heterogeneous MPI, 2006