

# How to install GAMMA: the Genoa Active Message MACHine

Giuseppe Ciaccio  
DISI, Università di Genova  
via Dodecaneso 35, I-16146 Genova, Italy  
ciaccio@disi.unige.it

2 May 2008

## 1 Introduction

### 1.1 About this document

This document is about installing the Genoa Active Message MACHine (GAMMA). It provides useful information about the following topics:

- the hardware/software components needed to use GAMMA (Section 2);
- configure, build and install the Linux 2.6 kernel for use with GAMMA (Section 4);
- download and configure the GAMMA source code (Sections 5 and 6);
- compile and install the GAMMA module (Section 7);
- set up a convenient environment for using GAMMA (Section 8);
- load and use GAMMA for running parallel jobs (Sections 9 and 10);
- possible causes and remedies for common troubles (Section 11).

This document is **not** about the GAMMA Application Programming Interface (API). The GAMMA API is described in the GAMMA home page on the WWW (<http://www.disi.unige.it/project/gamma/>), where you can also find information concerning GAMMA performance.

### 1.2 A sketch of the GAMMA software architecture

GAMMA is a messaging system designed for low-latency, high throughput inter-process communication across a Gigabit Ethernet LAN.

Different from many so-called “user-level” messaging system, GAMMA communications are (lightly) mediated by the Operating System (OS) of the local host computer; the GAMMA software architecture is therefore quite “classical”, being structured as three main components:

- a device driver, called the *GAMMA driver*, loaded as a module in the OS kernel, which implements basic communication routines;
- a set of so called “light-weight” system calls (a lower overhead implementation of the classical concept of OS system calls), which allow a user process, running in user space, to invoke the basic communication routines of the GAMMA driver; and
- a user programming library, called *GAMMA library*, which implements the GAMMA API on top of the “light-weight” system calls and provides user application programmers with point-to-point and (a few) collective communication routines.

GAMMA provides a non-standard set of communication routines. Programmers who prefer to write parallel applications using a more standard API might want to install MPI atop GAMMA (see the WWW page of the MPI/GAMMA project, <http://www.disi.unige.it/project/gamma/mpigamma>).

There is one GAMMA driver for each supported Network Interface Card (NIC) (see Section 2). Each GAMMA driver is a modified version of a standard Linux device drivers.

*NOTE:* within the GAMMA driver, the IP functionality is blocked. But GAMMA needs IP in order to start parallel jobs (the startup is done via `rsh` or `ssh` commands). Therefore, you need an additional LAN for the IP services. See Section 1.2.1 for details.

### 1.2.1 GAMMA and IP: the need for an additional LAN

The GAMMA drivers derive from standard Linux drivers. However, *GAMMA drivers do not support any IP-based protocols.*

This limitation allows better performance and stability. The drawback is that an additional network is needed, because GAMMA needs the IP services when starting the parallel job. The remote process instances are indeed generated by using the `rsh` or `ssh` services, so IP must work while launching the job. Moreover, the executable file is usually made available to all nodes via a network file system (NFS, for instance), which relies upon IP as well. So an additional network is necessary, where IP can run.

### 1.3 How to contact the GAMMA authors

The authors of GAMMA are:

- Giovanni Chiola, [chiola@disi.unige.it](mailto:chiola@disi.unige.it): Former project supervisor (do not contact him any more please);
- Giuseppe Ciaccio, [ciaccio@disi.unige.it](mailto:ciaccio@disi.unige.it): Current maintainer; and
- other persons who contributed new GAMMA drivers; see the GAMMA web page for a complete list, or look inside source code and read about authors.

## 2 Requirements

### 2.1 User prerequisites

In order to install GAMMA successfully, you should:

- be able to configure and compile a Linux kernel and install it on all cluster nodes, and
- hold root privileges on all cluster nodes.

### 2.2 Hardware/software requirements

The requirements to install GAMMA are:

- A pool of Personal Computers (PCs) with PCI bus.  
Supported CPUs are: Intel Pentium, AMD K6, and all superior models, including their 64-bit versions (so called “x86\_64” or “amd64” architecture, not to be confused with the IA-64).  
Multi-CPU nodes are allowed. It is not yet clear whether the current version of GAMMA is thread-safe from the user application standpoint, so you might have troubles in case of multi-threaded parallel jobs. Otherwise, the use of multi-CPU nodes is safe with GAMMA. Multi-core CPUs have not yet been tested with GAMMA, but they are expected to work fine.  
Hyperthreading seems to cause troubles, so it must be disabled by BIOS.

- At least one Gigabit Ethernet NIC on each node, of a kind supported by GAMMA. Currently, GAMMA supports the following (families of) NICs:
  - many adapters (possibly hardwired on the motherboard) equipped with one of the Intel PRO/1000 (8254x Gigabit Ethernet) chipsets;
  - many adapters (possibly hardwired on the motherboard) equipped with one of the Broadcom “Tigon3” Gigabit Ethernet chipsets.

You need at least one NIC on each node, of the kinds listed above. NICs not listed above may be present on nodes, but GAMMA will not use them.

- A Gigabit Ethernet switch, or a single (possibly crossover) cable for a “back-to-back” connection (two nodes only).
- At least one additional NIC on each node, and an additional hub/switch, to support the IP protocols, which GAMMA relies upon for spawning process instances (see Section 1.2.1). Such an additional NIC can be of whatever kind supported by the Linux kernel (including the ones supported by GAMMA itself).
- Linux kernel version 2.6.24 on all nodes.
- gcc version 4. Other compilers might also work fine, but we have not yet tested them.

### 2.3 Assumptions about file placement

For simplicity, we assume that the pathname of the Linux source tree is `/usr/src/linux/`. Other directories are allowed however, at the user choice.

## 3 Install GAMMA step by step

The installation procedure for GAMMA is not complex, but assumes you have some expertise in performing various tasks (see Section 2.1). By carefully reading the following Sections, you will be able to come to success after a number of steps that must be carried out in a precise order, namely:

1. Step 1: Configure, build and install the Linux kernel 2.6.24 on your cluster nodes (Section 4)
2. Step 2: Download and configure the GAMMA source code (Section 5)
3. Step 3 (optional): Tune some GAMMA parameters (Section 6)
4. Step 4: Compile and install the GAMMA module (Section 7)
5. Step 5: Set up the environment for GAMMA (Section 8)
6. Step 6: Load the GAMMA module on all cluster nodes (Section 9)
7. Step 7: Use GAMMA for running parallel jobs (Section 10)

The installation procedure can be simplified if you are installing a new release of GAMMA having already installed a previous one. In this case, if you are re-using the same Linux kernel and no changes to the cluster environment are planned, you may skip steps 1, 5, and 6.

Most of the above steps require root privileges, so be very careful.

## 4 Step 1: Configure, build and install the Linux kernel 2.6.24

Before installing GAMMA, it is always necessary to configure and build the Linux 2.6 kernel in a proper way. So, our first step is to configure the Linux kernel.

Enter directory `/usr/src/linux/` (or whatever directory of your Linux source tree), and type:

```
make menuconfig
```

or:

```
make config
```

If you are upgrading from a Linux kernel older than 2.6.24, you may save time and recycle your previous kernel settings. Copy the `.config` file from your previous kernel tree to the current one, then type:

```
make oldconfig
```

and set up only the additional features of your new kernel.

Whatever your choice, a script (which you should be familiar with) is started which will prompt you to answer a number of questions concerning your Linux kernel configuration.

### 4.1 Linux kernel settings required by GAMMA

In section `Processor type and features`, the feature

`Preemption model`

*must be* set to `No Forced Preemption`.

You *must also disable* the feature

`Preempt the Big Kernel Lock`

In the same section, if you have multiple-CPU nodes (or multicore CPUs) you might want to enable the feature

`Symmetric multi-processing support`.

Under section `Kernel hacking`, you *must disable* the feature:

`Use 4Kb for kernel stacks instead of 8Kb`

if present.

### 4.2 Selecting network device drivers

Under section `Device Drivers`, subsection `Network device support`, you should enable the drivers for each NIC supposedly present on the cluster nodes. You may include drivers also for those NICs that are expected to be operated by GAMMA; the GAMMA driver module, once loaded, will supersede these drivers in a safe way.

### 4.3 Build and install the kernel

Build the kernel in your favourite way (usually `make` && `make install` && `make modules_install`, but there might be exceptions). When done, you might look under directory `/boot` and see a fresh `vmlinuz` kernel and a fresh `System.map` file, plus kernel modules under directory `/lib/modules/2.6.24`.

In order to install the new kernel on all cluster nodes, copy the fresh `vmlinuz` and `System.map` files on all nodes; *do not omit* `System.map`, as this file is a key ingredient for later use of the GAMMA driver. A typical directory for `vmlinuz` and `System.map` is `/boot` on all nodes, but diskless nodes may require different settings. You should also put a copy of the entire module directory `/lib/modules/2.6.24` on all nodes.

After the new kernel has been installed, you might need to edit your bootloader configuration file (`/boot/grub/menu.lst`, `/etc/lilo.conf`, ...) on each node, so that your bootloader will take your new kernel into account at next reboot. If LILO is your bootloader, please remember to run `lilo` after having edited `/etc/lilo.conf`.

### 4.4 Reboot your cluster

Reboot your cluster nodes to see if your Linux 2.6.24 kernel has been configured and installed correctly. You may safely want to first reboot just one node and see what happens; if everything is fine, go ahead and reboot every other node.

## 5 Step 2: Download and configure the GAMMA source code

### 5.1 Download the GAMMA package

Go to the GAMMA WWW page, <http://www.disi.unige.it/project/gamma/>, section “Software”, and click on the link pointing to the GAMMA package; your browser will prompt you to download (by HTTP) a file named like “gamma-YY-MM-DD.tar.gz” (for instance: `gamma-06-02-17.tar.gz`). Get it.

For simplicity, we will call the GAMMA package `gamma.tar.gz`, that is, we will omit the “YY-MM-DD” part of the file name (which changes every time a new release of GAMMA is put on the WWW).

### 5.2 Unpack the GAMMA package

Place `gamma.tar.gz` into your favourite working directory. Now, invoke this command:

```
tar xvzf gamma.tar.gz
```

This operation will create a subdirectory named `gamma`, containing the GAMMA source code.

### 5.3 Configure the GAMMA source code

Enter the subdirectory `gamma/` and run the script named `configure`. This script will ask you a few questions, concerning some configuration parameters of your local installation of GAMMA. The possible answers, along with the default answer, are indicated as well. In the remaining part of this Section, we explain the meaning of each question of the `configure` script.

Select the CPU architecture [`"i386"`, `"x86_64"`]

Specify the CPU architecture of your cluster’s processors. Please note that the architecture “`x86_64`”, sometimes referred to as “`amd64`” (Opteron, Athlon64) has nothing to do with the IA64 architecture (Itanium). The architecture “`i386`”, however, refers to the classical IA32 (Pentium, AMD Athlon 32 bit, etc.). In response to this question, the script creates a number of symbolic links to the appropriate files containing source code that is dependent on the CPU architecture.

Select the Network Interface Card (NIC) to be operated by GAMMA

Specify which type of NIC (Intel PRO/1000, Tigon3) you are going to operate with GAMMA. In response to this question, the script creates a number of symbolic links to the appropriate files containing source code that is dependent on the specific NIC.

How many [Intel PRO/1000 | Tigon3] NICs are to be operated by GAMMA on each node

Specify how many NICs of the afore-specified type are to be operated by GAMMA on each cluster node. Default is 1. The remaining NICs will be operated by standard Linux kernel drivers.

If the selected NIC was an Intel PRO/1000, and GAMMA is to operate more than one NIC on each node, the following additional question will be asked by the configure script:

Enable support for Flat Neighbourhood Networks (FNN)  
(FNN) [experimental] (y/n)

A Flat Neighbourhood Network (FNN) (<http://www.aggregate.org/FNN/>) is an interconnection scheme for clusters. Usually, in a N-nodes cluster, each node has just one NIC for fast communication, and a single N-way switch connects all nodes in a star-shaped topology. With FNN each node must have more than one NIC; there are more than one switch, and each node is connected to more than one switch in such a way that the path from any two nodes traverses at most one switch. The topology is equivalent to a N-way star, but made of smaller switches instead of a big, single one. It should cost less. Of course each node should have more than one NIC operated by GAMMA. This feature is still experimental and not fully tested.

Enable Jumbo Frames (USE\_JUMBO\_FRAMES) (y/n)

Most Gigabit Ethernet NICs support so called *Jumbo Frames*, that is, they allow packets of greater size than the Ethernet standard. This feature, which also must be supported by the LAN switch, might yield fairly greater communication throughput. You should however turn it off if the LAN switch does not support Jumbo Frames, or if you are unsure about this. This feature is disabled by default.

*NOTE:* some Gigabit Ethernet adapters do not support Jumbo Frames at all.

*NOTE: Intel PRO/1000:* some Intel PRO/1000 adapters (8254x chipsets) may become unreliable when Jumbo Frames are enabled, with an increased probability for the NIC to hang-up during use. The GAMMA driver for the Intel PRO/1000 can detect and solve these hang-ups, but performance may suffer; in case of frequent hang-ups, you should consider disabling this feature. See end of Section 11 for details.

Allow more process instances of same job on same node  
(recommended with multi-CPU nodes)  
(MORE\_INSTANCES\_ON\_SAME\_NODE) (y/n)

Each parallel job is made up of a number of cooperating processes, called *process instances*. For best performance, each PC should run as many process instances as the CPUs available locally, so as to allow all instances to be simultaneously running. In case you wish to run more than one process instance on the same node (for instance, because the node is a dual-CPU), you should enable the “more process instances on same node” feature. Please note that this has *nothing to do with time-sharing the cluster among more parallel job* (a feature GAMMA always

supports by default).

How many process instances per node  
(recommended answer: same as number of CPUs per node)  
(MAX\_NUM\_LOCAL\_INST)

If support for multiple instances on same node has been enabled, here it is possible to specify how many process instances are to be run on each node. It makes much sense that this be equal to the number of CPUs available per node.

Strip away GAMMA-related args at job launch (STRIPARGS) (y/n)

This option is to be considered if you want to use MPI atop GAMMA and your MPI job refuses to start (a notable example is NetPipe). GAMMA uses inline args to send some information to remote process instances when spawning them. These args can cause some picky MPI programs (e.g. NetPipe) to abort during initial phases. The n answer (default) is usually fine, but if your NetPipe for MPI then refuses to start, you have to change to y and recompile your MPI/GAMMA and Netpipe. With this option enabled, however, those MPI applications that look at the args before calling `MPI_Init()` might show startup problems; notable examples are all the NAS benchmarks. So, don't enable this option unless your own MPI programs really need it.

Use flow control in GAMMA barrier sync (USE\_FLOWCTL\_IN\_SYNC) (y/n)

Specify whether to use flow control when performing a GAMMA barrier synchronization. Given the synchronization semantics of such a collective routine, flow control is usually unnecessary (default).

Print info when launching GAMMA jobs (VERBOSE) (y/n)

Specify whether to enable the printout of additional information at the time of launching a GAMMA job.

Spin-yield instead of busy-waiting on receive (SPIN\_YIELD) (y/n)

Specify whether to use spin-yield mode instead of pure busy-waiting whenever waiting for message receive. The default answer is y, and is strongly recommended especially if the cluster is used in non-dedicated mode.

Use ssh in place of rsh to spawn remote process instances (USE\_SSH) (y/n)

Specify whether to use ssh for spawning processes on remote nodes when launching parallel jobs. As an alternative, rsh can be used to this end. Default is ssh.

Where is the Linux source tree installed?

Specify the pathname of the Linux source tree on the machine where you are going to compile the GAMMA-enhanced Linux kernel (default `/usr/src/linux/`).

Where would you like to install the GAMMA user library?

Specify the directory where to install the GAMMA user library, once compiled (default `/usr/lib/`).

Where would you like to install the GAMMA startup/recovery utils?

Specify the directory where to install various utilities for GAMMA startup and recovery, including the GAMMA module `gamma.ko` (default `/usr/local/bin/`).

## 6 Step 3: Tune some GAMMA parameters (optional)

This Section is concerned about some settings of GAMMA which are not managed by the `configure` script discussed in Section 5.3).

### 6.1 NIC-specific settings

All GAMMA parameters whose value depends on the particular NIC in use are defined as constants in file `gamma_nic_dependent.h` (which actually is a symbolic link created by the `configure` script, see Section 5.3).

The default settings for such constants should ensure a correct behaviour of GAMMA on all platforms, so you usually do not need change anything here.

#### 6.1.1 GAMMA\_TX\_RING\_SIZE and GAMMA\_RX\_RING\_SIZE

A couple of constants the value of which you might want to change are `GAMMA_TX_RING_SIZE` and `GAMMA_RX_RING_SIZE`.

Increasing the value of `GAMMA_RX_RING_SIZE` may lead to slightly better performance with GAMMA flow-controlled communication, but will consume more memory at kernel level in the OS; conversely, decreasing the value might be necessary if your PCs are short of RAM (less than 32 MBytes), but performance of GAMMA flow-controlled communication could decrease slightly (to know more about why the setting of `GAMMA_RX_RING_SIZE` may affect performance of GAMMA flow-controlled communication, see Section 6.1.2).

Increasing the value of `GAMMA_TX_RING_SIZE` may lead to better performance with GAMMA non-blocking transmission, but will consume more memory at kernel level in the OS; conversely, decreasing the value might be necessary if your PCs are short of RAM, but performance of GAMMA non-blocking transmission could suffer.

We suggest you *not* to increase the default settings, because the performance improvement obtained in this way is quite negligible. We also suggest you not to decrease the default settings unless your PCs are really short of RAM.

#### 6.1.2 GAMMA flow control

GAMMA provides both best-effort and flow-controlled communication routines. GAMMA flow control prevents overrun at the receiver NICs from occurring, therefore avoiding packet losses at the end stations (this does not completely solve the problem of packet losses in a LAN, as they might also be caused by switch congestion).

GAMMA flow control is based on *credits*. A credit is an amount of packets that a sender station is guaranteed to be able to deliver to a receiver station safely, that is, without causing a receiver overrun. If station S1 has a credit of  $C$  towards station S2, then S1 can safely send  $C$  packets to S2 at full speed, and  $C$  is decremented by one each time a packet is transmitted; when  $C$  is zero S1 has to stop, waiting for a “credit renewal” from S2. The larger the initial value of  $C$ , the faster the communication from S1 to S2; however, too large an initial value for  $C$  might cause an overrun to S2.

The parameter `MAX_CREDIT_SIZE` defines the maximum size of credit of each station in the cluster towards each other station. Another parameter, namely `LOW_WATERMARK`, determines the amount of remaining credit



below which a request for credit renewal is issued. The maximum values allowed for these two parameters are subject to constraints: The latter must not be greater than the former, and their sum should never exceed the value  $(\text{GAMMA\_RX\_RING\_SIZE} - 20) / (\text{total\_num\_nodes} - 1)$ , which is nearly the maximum value ensuring that a receiver NIC will not overrun when simultaneously flooded by all possible senders with GAMMA flow-controlled communications (a very conservative setting; actually, this constraint can be released, and indeed must be released when running many nodes).

From the above, it is clear that the larger `GAMMA_RX_RING_SIZE`, the larger the maximum value for `MAX_CREDIT_SIZE`, which translates into slightly better performance of GAMMA flow-controlled communication (as pointed out in Section 6.1.1).

Anyway, we suggest you do not change the default settings for `MAX_CREDIT_SIZE` and `LOW_WATERMARK`.

## 6.2 Protocol-specific settings

Here we examine some GAMMA parameters which affect the behaviour of GAMMA independent of the particular NIC in use. These are defined as constants in file `gamma_def.h`. A common user will leave these settings untouched.

### 6.2.1 Maximum cluster size

The maximum allowed cluster size (number of PCs allowed to connect to the cluster LAN) is defined by the constant `MAX_NUM_NODES` in file `lib/gamma_userlev.h`; this is currently set to 128. Some internal data structures of GAMMA need an amount of statically allocated memory which is proportional to the maximum cluster size. Therefore, in case the PCs of your cluster are short of RAM, the default value may cause a failure during boot due to shortage of available memory. Should this ever occur, or even to prevent this from occurring, the default value for `MAX_NUM_NODES` can be set to a smaller value (e.g., 16). Of course, this imposes a limit on the effective size of your cluster. If, on the other hand, you wish to install GAMMA on a cluster with more than 128 nodes, set `MAX_NUM_NODES` to a larger value, *not exceeding 254*. Note that this defines the maximum number of nodes, not CPUs.

### 6.2.2 Debug kernel messages

GAMMA contains a number of statements which generate debug-oriented messages as well as warnings about communication failures. The constants `DRIVER_DEBUG_EVERYTHING` and `DRIVER_DEBUG` govern the conditional compilation of commands producing debug-oriented messages, whereas the constant `DRIVER_DEBUG_LIGHT` governs the conditional compilation of commands producing failure warnings. By default, the first two constants are “undefined” (`#undef DRIVER_DEBUG_EVERYTHING` and `#undef DRIVER_DEBUG`), whereas the third one is “defined” (`#define DRIVER_DEBUG_LIGHT`); changing the second constant to “defined” enables the compilation of code for the debug-oriented messages, and changing the first one to “defined” enables even more such messages; conversely, changing the third constant to “undefined” removes the code for emitting failure messages.

## 7 Step 4: Compile and install the GAMMA module

After configuration and tuning, the GAMMA module can be compiled and installed. Enter subdirectory `gamma/` and run the command

```
make
```

Ignore the (possibly many) warnings that might show up during compilation. Next, install the GAMMA module by invoking

```
make install
```

The GAMMA module is called `gamma.ko` and is put under the install directory specified at configure time (usually `/usr/local/bin`).

## 8 Step 5: Set up the environment

### 8.1 User-specific environment

Most GAMMA utilities, including the startup of parallel jobs, need `ssh` in order to work across nodes. Therefore, you have to configure and enable `ssh` on all of your cluster nodes to work passwordless (that is, in a way such that commands can be launched remotely without repeatedly asking for the user password). This must be done on a per-user basis. Commands like `ssh-keygen` will be of help; ask a web search engine or a system administrator if you don't know how to do.

The shell variable `PWD` must be present in each user environment. For instance, users who prefer the bash shell should add the following statement to their `.bash_profile` file:

```
export PWD
```

To launch a GAMMA parallel job, one copy of the executable must be present on each cluster node and all the copies must have the same absolute pathname and filename. The obvious way to obtain this, is to have one single copy of the executable and let all nodes share that executable via NFS; to this end, the cluster administrator might have to act upon files `/etc/fstab` and `/etc/export` on all nodes. Please note that in a large cluster this might be impractical (when starting the parallel job, a huge contention would put the NFS server into troubles).

### 8.2 Cluster-specific environment

#### 8.2.1 Hyperthreading must be disabled by BIOS

Do not forget to disable Hyperthreading by BIOS, on each node. GAMMA suffers some instabilities when Hyperthreading is enabled.

#### 8.2.2 Put GAMMA module and startup/recovery utilities on all nodes

Let us suppose you have chosen `/usr/local/bin/` as the directory where the GAMMA startup/recovery utilities are installed.

Enter directory `/usr/local/bin/`; there, you should find the GAMMA module (`gamma.ko`) along with a number of scripts and executables (the GAMMA startup/recovery utilities). All these files have in common the `gamma` prefix in their names. Copy these files to all your cluster nodes, again under `/usr/local/bin/`.

### 8.3 Generate the GAMMA configuration file(s)

GAMMA needs a description file called `/etc/gamma.global`, defining the cluster topology.

The most frequent case is a star-shaped topology: `N` nodes, one NIC each, connected to a single switch. Here is a sample `gamma.global` file for a cluster with four nodes (resp. named “felix”, “orion”, “aries”, and “libra”) and a star topology. In this example, the NIC operated by GAMMA is `eth1` for all nodes. This is a frequent case, since `eth0` is usually devoted to IP traffic.

```
subnet {
felix eth1
orion eth1
aries eth1
libra eth1
```

```
}
```

*Important Notice:* you must know in advance (or at least figure out) which of your `eth` devices shall be driven by GAMMA on each node, in order to write a correct `gamma.global` file. In case you do not have such information, the only way to get it is to load the GAMMA module (see Section 10 without any `gamma.global` file; GAMMA will not work, but then you may invoke the `dmesg` command from a shell and look at the output, in order to see which of the `eth` devices has been taken over by the GAMMA driver.

The host names reported into `/etc/gamma.global` should be the ones returned by command `hostname` on each node. Comments are allowed in the file, prepended by a `#`. Blanks and tabs are allowed as well.

You have to prepare a single `/etc/gamma.global` file on just one node of your cluster (usually the front-end, but another node would also do). There is no need to put it on other nodes.

Things are a bit more complicated when the cluster topology is a Flat Neighbourhood Network (FNN). The starting point is again a high-level description file `/etc/gamma.global`, but the syntax for FNNs is slightly different: There are more than one switch (subnet), and each node has more than one NIC (“eth1”, “eth2”, ...; usually “eth0” is for IP). Here is a sample `gamma.global` file for a FNN connecting twelve nodes (named “bin01”, “bin02”, and so on) with three 8-way switches:

```
subnet {
bin01 eth1
bin02 eth1
bin03 eth1
bin04 eth1
bin05 eth1
bin06 eth1
bin07 eth1
bin08 eth1
}
```

```
subnet {
bin01 eth2
bin02 eth2
bin03 eth2
bin04 eth2
bin09 eth1
bin10 eth1
bin11 eth1
bin12 eth1
}
```

```
subnet {
bin05 eth2
bin06 eth2
bin07 eth2
bin08 eth2
bin09 eth2
bin10 eth2
bin11 eth2
bin12 eth2
}
```

Other sample `gamma.global` files can be found under directory `gamma/util/` in the GAMMA source pack-

age.

Also with FNNs you just need a single `/etc/gamma.global` file on just one cluster node.

After having emplaced `/etc/gamma.global`, the next step is to switch all nodes on, then run the utility program `gammamkconfall` on the same node where `/etc/gamma.global` has been emplaced. `gammamkconfall` has to be launched just once, but uses `ssh` to collect information from all other nodes, so all nodes must be switched on and `ssh` must work across them before doing this step.

After completion of `gammamkconfall`, all nodes have been properly set up and are ready for loading the GAMMA module.

`gammamkconfall` must be run again whenever the cluster composition changes. Report the changes into the `gamma.global` file, then run `gammamkconfall` again (with all nodes switched on).

## 9 Step 6: Load the GAMMA module on all cluster nodes

In order to load the GAMMA module on all cluster nodes you have two possibilities, namely:

- run the script `gammainsmodall` from just one node (usually the front-end)
- or, run the script `gammainsmod` on each individual node.

The former is more suitable if you just want to load the GAMMA module when you need it, or when playing with GAMMA the first time. It has the advantage that you just launch it on one node (it will use `ssh` to force module loading on all other nodes).

The latter is preferred when you want the GAMMA module to be loaded automatically at boot time; in this case, put the invocation of `gammainsmod` in the appropriate boot-time script on all nodes (depending upon your Linux distribution it can be `/etc/rc.d/rc.local` or `/etc/conf.d/local.start`, or maybe somewhere else).

In any case, *never load the GAMMA module with `insmod` or `modprobe`!* It will not work, because the GAMMA module needs appropriate values for some parameters, and these cannot be easily provided by hand.

After completion of `gammainsmod` or `gammainsmodall`, check the boot messages on all nodes, using the command `dmesg`. If you see a message like:

```
GAMMA Genoa Active Message MACHine (Linux 2.6)
```

followed by a calendar date and other information, then the GAMMA module has been loaded successfully and the NICs have been probed.

### 9.1 No need to configure the `eth` devices by hand

The `gammainsmod` or `gammainsmodall` scripts automatically configure the `eth` devices corresponding to the NICs in use by GAMMA, so you don't have to run any `ifconfig` command by hand.

### 9.2 NIC “predation” by the GAMMA module

It may be the case that the NICs you are going to use with GAMMA are by default registered with their own original Linux driver. You don't have to worry about this: the GAMMA module “predates” the NICs it needs and superseeds any possibly preexisting driver. When the GAMMA module is unloaded, the original driver is put back in place.

### 9.3 Unloading the GAMMA module

To unload the GAMMA module, do `rmmod gamma` on each node or run the script `gammarmmodall` on just one node.

Do not try to run any GAMMA parallel job when the module is unloaded: it will not work anymore.

When the GAMMA module is unloaded, any previously predated NIC is given back to its original Linux driver. However, the `eth` devices corresponding to the predated NICs are left disabled, so you (or the Linux hotplug logics, if properly configured) will have to bring the devices up, should you need them. Use the `ifconfig` command.

## 10 Step 7: Use GAMMA for running parallel jobs

### 10.1 Running a GAMMA parallel job written in C

As already pointed out in Section 8.1, in order to launch a GAMMA application, one copy of the executable must be present on each cluster node and all the copies must have the same absolute file name. The best way to obtain this, is to have one single copy of the executable let all nodes to share it via NFS. Make also sure the shell variable `PWD` is exported in the user environment.

To launch a GAMMA parallel job, simply type the name of the executable followed by the in-line arguments required by the program itself. For instance, to launch a GAMMA parallel program whose name is `ping_pong` and requiring one numeric in-line argument with value 5, simply type

```
ping_pong 5
```

Note that no indication is given as for the degree of parallelism. With GAMMA, the number of process instances to spawn must be explicitly set up by the program itself once launched (of course the program can always get the desired degree of parallelism from the user through an in-line argument).

In the above example, no indication is given concerning which nodes the parallel job should use. By default, the first process instance (with instance number 0) always runs on the local machine (the one the job has been launched from); the names of other nodes needed for other process instances are taken from file `/etc/gamma.conf` (generated by `gammamkconfall`), by reading the file circularly starting from the name just after the one of the local machine (with repetition).

As an alternative, you may designate explicitly which nodes are to run the job. For instance, let us suppose the user wishes to run a GAMMA job called `barrier` with four process instances, using the two machines `libra` and `orion` so that instances 0 and 1 run on `libra` and instances 2 and 3 on `orion`. To this end, the user must first create a *machine file* like this:

```
libra
orion
orion
```

then must log on `libra` and launch the job from that machine. If the name of the above machine file is `mach`, to launch the job the user should type:

```
barrier 4 -machinefile mach
```

The GAMMA runtime support will then spawn the process instances so that the first one (instance 0) runs on the local machine (`libra`), whereas the other three instances run on the machines listed in the machine file, in the same order.

### 10.2 Sample GAMMA parallel C programs

Under directory `gamma/apps/pingpong/` you can find a few simple GAMMA parallel jobs for latency and bandwidth measurements, which can also be used for testing the functionality of GAMMA. A Makefile is provided to compile them.

The most important of such programs is `ping_pong.c`. The executable `ping_pong` performs a number of message round-trips in order to measure the average GAMMA communication delay between two nodes. To run it, you must simply invoke `ping_pong` followed by an argument, namely, the size in bytes of the message to be exchanged. If size is zero, then the program will output the one-way latency time (half the round-trip time) in  $\mu\text{sec}$ ; otherwise, it will output the one-way throughput in MByte/s, computed as the message size divided by half the average round-trip time. The source code of `ping_pong` contains a number of options, in the form of C constants that can be “defined” or “undefined” at will; for instance, you can choose the specific GAMMA communication routine to be used for the test (best-effort, flow-controlled, etc.) by acting over constants `USE_2P`, `USE_ISEND`, `USE_FLOWCTL`.

Another simple but interesting GAMMA parallel program is `barrier.c`, under `gamma/apps/barr/` (there is also a Makefile for building the executable). Once compiled, the executable `barrier` takes one argument, namely, a number `N`, and measures the average barrier synchronization time among `N` processes in the cluster. `N` is not allowed to exceed the total no. of nodes in the cluster multiplied by the maximum number of process instances allowed per node.

For other GAMMA applications, explore the directory `gamma/apps/`. Please note, most of these sample programs are very old and not maintained at all.

### 10.3 Running GAMMA parallel jobs written in FORTRAN

Under `gamma/apps/pingpong/fortran/` you can find a sample “ping-pong” GAMMA parallel program written in FORTRAN, plus a Makefile to compile it. Basically, this program is there only to test if it can compile and run.

To launch a parallel job written in FORTRAN, you must use the utility `gammarun`. In our example, type:

```
gammarun 2 ping_pong
```

where the “2” specifies the number of computing nodes involved in the application run.

### 10.4 Restarting the network after a crash

GAMMA is still far from being a bullet-proof communication system. If something goes wrong during a run of a GAMMA parallel application, and your network appears to be no longer working, you might try to recovery by first killing your job, then run the script `gammaresetall` on just one node.

Even better, run the script `gammacreditall -r` on just one node.

### 10.5 Adjusting the size of GAMMA flow control credits

In some cases, it might be helpful to act upon the size of the credits in the GAMMA credit-based flow control (see Sections `reffield`).

To adjust the credit size, run the script

```
gammacreditall <new_credit_size> on just one node in the cluster.
```

For example, by typing:

```
gammacreditall 12
```

the credit size is set to 12 packets on all nodes.

To restore the credit size to the default value, type:

```
gammacreditall -r
```

To read the current credit size, simply type:

gammacreditall

## 10.6 Adjusting the timeout for missing packets in the GAMMA retransmission algorithm

GAMMA implements mechanisms to detect and possibly retransmit missing packets. To allow detecting packet losses, each GAMMA packet travelling from node  $A$  to node  $B$  is tagged by a unique id number  $Id(A, B)$ . The current value of  $Id(A, B)$  is incremented by one at each packet transmission from  $a$  to  $B$ ; thus, any node can detect packet losses by simply checking the sequences of id numbers tagging its incoming packets.

This mechanism works enough in most cases; there is an exception, however, namely: if a packet get lost, and no more packets are transmitted, the missing id number cannot be detected. It is therefore necessary to run a mechanism based on a *timeout* at the receiver side, in order to catch all possible packet losses.

In some cases, it might be helpful to act upon the duration of the timeout. Too short a timeout, indeed, might force unneeded packet retransmissions; on the contrary, too long a timeout turns out into slower recovery from packet losses.

To tune the duration of the timeout, you can run the script `gammamaxpollsall` on just one cluster node. For example, by typing:

```
gammamaxpollsall 5000
```

the timeout is set to approx. 1 sec on all nodes.

To restore the timeout to the default value, type:

```
gammamaxpollsall -r
```

To read the current setting for the timeout, simply type:

```
gammamaxpollsall
```

## 11 Quick troubleshooting

This is a list of common troubles that you might encounter when using GAMMA. For each symptom there is a tentative diagnosis and one or more suggestions for actions that could help solve the problem.

This Section is not to be intended as complete. Future versions of this document might hopefully add more paragraphs to this Section, also with the help of GAMMA users.

**Symptom** You try to launch a GAMMA application but nothing seems to happen.

**Suggestions** It may depend on many causes. If GAMMA was configured to use Jumbo Frames, check whether the NICs and the LAN do support Jumbo Frames. Check whether the GAMMA driver was able to detect and probe the NIC (Section 9). Check whether the GAMMA configuration file `/etc/gamma.conf` (Section 8.3) exists on all cluster nodes. If not, run `gammamkconfall` (Section 8.3). Check whether the executable file of the GAMMA application is seen on all the involved nodes with the same absolute pathname. In doubt, recompile the GAMMA library with “verbose” mode turned on, then recompile the application and try again.

**Symptom** You try to launch a GAMMA application and the following message appears on the standard output:

```
gamma_init(): Could not read /etc/gamma.conf
```

**Suggestions** The GAMMA configuration file `/etc/gamma.conf` could not be read on one or more nodes in the cluster. Run `gammamkconfall` (Section 8.3).

**Symptom** You try to launch a GAMMA application and the following message appears on the standard output:

```
gamma_init(): could not create a virtual GAMMA
```

**Suggestions** GAMMA failed to create some local runtime data structures. The most likely reason for this, is that many GAMMA applications were badly killed before this attempt. Invoke the utility `gammacreditall -r` (Section 10.4), then try again. If GAMMA was configured to use Jumbo Frames, check that the NICs and the LAN do support Jumbo Frames.

**Symptom** You try to launch a GAMMA application and the following message appears on the standard output:

```
gamma_init(): registration failed
```

**Suggestions** GAMMA failed to create some runtime data structures across the cluster. This might indicate a communication problem on the LAN. Check whether the GAMMA driver was able to detect and probe the NIC (Section 9). If GAMMA was configured to use Jumbo Frames, check that the NICs and the LAN do support Jumbo Frames. Check whether the GAMMA configuration file `/etc/gamma.conf` (Section 8.3) exists on all cluster nodes. If not, run `gammamkconfall` (Section 8.3).

**Symptom** You try to launch a GAMMA application and the following message (or a very similar one) appears on the standard output:

```
gamma_init(): launch on node 3 failed!
```

**Suggestions** GAMMA failed to spawn one or more remote processes of the parallel application. Check whether you can run commands on remote nodes via `ssh` (Section 8.1). Check whether the shell variable `PWD` is exported in the user environment (Section 8.1). Check whether the executable file is seen on all the involved PCs with the same absolute pathname (Section 8.1).



**Symptom** You try to launch a GAMMA application and the following message (or a very similar one) appears on the standard output:

```
bash: /root/pingpong/pingpong: No such file or directory
```

**Suggestions** Check whether the executable file is seen on all the involved PCs with the same absolute pathname (Section 8.1).

**Symptom** You try to launch a GAMMA application and the following message (or a very similar one) appears on the standard output:

```
bash: syntax error near unexpected token `(null)/pingpong'
bash: -c: line 1: `(null)/pingpong 0 -GAMMAHOME (null)/ -GAMMA 1'
```

**Suggestions** Check whether the shell variable PWD is exported in the user environment (Section 8.1).

**Symptom** A GAMMA job freezes during run, and the `dmesg` command on all or some processing nodes shows messages similar to these ones:

```
sys_gamma_send(): par_pid 1, out_port 1, prog_no 3: Failure
sys_gamma_send(): par_pid 1, out_port 1, prog_no 3: Fail after 10 trials
```

**Suggestions** This usually indicates a LAN hardware problem; check your LAN hardware (especially the cables). It might also indicate a very congested LAN, where a transmitting NIC could not send packets for too long a time.

**Symptom** During execution of a GAMMA parallel job, `dmesg` command on all or some processing nodes shows the following message, repeated a number of times:

```
sys_gamma_poll(): no pkts for a long time. Testing frame seq # and credits...
```

**Suggestions** This might indicate three very different conditions, namely:

- The duration of the retransmission timeout (Section 10.6) is too short. This forces numerous yet unneeded checks for missing packets. Use the `gammamaxpollsall` utility (see Section 10.6) to decrease the value of the timeout.
- The running parallel job congests the LAN switch, and therefore a lot of packets get lost. This might result into slower completion of the job. You probably need a different, more performing but possibly more expensive interconnect; as an alternative, you might try to modify the parallel program in order to enforce a better communication pattern among cooperating processes of the parallel job.

- A process instance A of your parallel job expects messages from some other process instance B, but B is not transmitting (because of reasons depending on your application). After waiting for a while, A will probe the network for possible loss of messages. You can ignore this normal behaviour, unless you see that your job does not make progress for too long a time.

**Symptom** The GAMMA test program `ping_pong` works with messages of size below 1500 bytes, while does not work with larger messages.

**Suggestions** You configured GAMMA to use Jumbo Frames (see Section 5.3), but your NIC and/or switch does not support them, or the MTU size for the network device is not set to the proper value. In the former case, configure GAMMA again and make sure the Jumbo Frames feature is disabled.

**Symptom** Your parallel job experiences occasional slow-downs or temporary hang-ups, and the following message frequently appears in the output of the `dmesg` command at some nodes of your cluster:

```
tx_flush_ring(): NIC restarted
```

**Suggestions** You configured GAMMA to use Jumbo Frames with the Intel PRO/1000 NICs (see Section 5.3). These NICs may become unreliable when Jumbo Frames are enabled. The probability of hang-up increases if your parallel job is such that each node sends messages to numerous other nodes, or the MTU size is greatly larger than the standard 1500 bytes. If you experience this kind of problem, configure GAMMA again and disable the Jumbo Frames feature.