NSM2 — Network Shared Memory

User's manual (for alpha version 1.9.xx)

Mikihiko Nakao, KEK

March 1, 2013

1 Introduction

Network shared memory (NSM) version 2 is a software package which provides two basic information exchange mechanisms over a TCP/IP based local area network. The first one is the **NSM distributed shared memory**, a mechanism to synchronize a given memory space by UDP broadcast over various hosts on a network segment. The second one is called **NSM message**, a mechanism to send a TCP packet in a user program without any knowledge of TCP/IP mechanism or system calls, host addresses or network configurations.

The main purpose of the NSM package is to control and monitor a data acquistion system of a large scale high energy physics experiment, but it is not necessarily limited to this field. The target user is a physicist who can write a simple sequential C or C++ program, not a computing engineer who may enjoy a sleepless debugging night of a deep-nested class objects. A typical data acquisition system of such an experiment consists of 10 to 1000 sub-systems and these sub-systems are required to work together by communicating with each other for the safe and stable operation. NSM version 1 was developed for the Belle experiment and has been used successfully for more than 10 years, and it was also used in several other experiments.

Then why NSM version 2? There have been certainly many lessons learned at the Belle experiment and the start of the Belle II experiment is an excellent opportunity to refurbish the package. NSM version 2 supports Linux and other Unix(-like) systems, but so far tested only on Linux systems with 32-bit and 64-bit kernels. Unlike the days of NSM version 1, there are not many choices of an operating system to test so far, but it is still designed to work on both little- and big-endian byte ordering with POSIX system calls. It can now properly run on a 64-bit operating system as well as a 32-bit one, and should survive year 2038 when the 32-bit unix time will overflow. On the other hand, some of the operating systems will no longer be supported. In particular, VxWorks real-time operating system is out of support. It may not work on some of the ancient operating systems. So far it is not decided to support operating systems such as Windows, iOS, and Android.

Setting up an **NSM network** is easy. First, one has to run an **NSM daemon (nsmd2)** on each host. This daemon takes care of all network communications over network, including nasty error handling, instead of you. The daemons automatically talk each other to build up a network, so you don't need to provide any configuration file or database. You may wish a little bit more of control over the daemons, which will be described later.

Then you need to write a standalone **user program** to talk to the NSM daemon, **but don't worry**. There are simple **example programs**, and there are less than 10 functions that have to be learned. The program has to be linked to the **NSM libraries**. There are two library levels, one is the **core** library and the other is a **tailored** library for the Belle II experiment. A user has to call only the functions in the tailored library. The tailored library includes predefined names and is expected to be modified for different purposes.

In a user program, a distributed shared memory has a user defined data type (**struct**), which is defined by the user and written down in an include file. Then information written to the struct will be shared and can be accessed by other programs who includes the same include file that defines the data type, with some small delay over the network. The use of NSM message is also simple. A program registers a callback function, in order to perform some action for the message. Then the program can continue its own job or just sleep until somebody else sends the message. When it is received, the callback function is called asynchronously. Sending a message is done by one line of code from a user program.

2 Quick Start

If NSM2 is not installed on your system, you can easily compile it from the distribution kit. The client program requires the GNU readline library development package, which may not be installed on your system. (Required package is “readline-devel” Scientific Linux 6 and it may be the same for RedHat/Centos; it is “libreadline6-dev” for Ubuntu 10.4 and it may be the same or similar for other Ubuntu/Debian Linux.)

Following four steps usually work on the supported platforms.

```
% gzip -d < nsm2-alpha.tar.gz | tar xf -
% cd nsm2-alpha
% ( cd daemon  ; make )
% ( cd corelib ; make )
% ( cd b2lib   ; make )
% ( cd example ; make )
```

(this procedure will be improved in the future releases.)

If you don’t like to use readline, you can instead try

```
% ( cd example ; make -f Makefile.no-readline )
```

Then, if NSM daemon `nsmd2` is not running on your system, you have to run it.

```
% daemon/nsmd2
```

This will run the daemon in foreground. If you feel safe, you can also run it in background by adding `-b` option.

There are four example clients in the `example` directory. You can run `master`, `client` and `readdat` in three windows. They can be on the same host or different ones as long as `nsmd2` is running. Each program requires a unique nodename which you have to provide.

```
terminal-1\% ./master master
  :
master>

terminal-2\% ./client test01

terminal-3\% ./readdat reader test01
```

The `master` program is your control terminal, from which you can type in control messages. The client program is a remote program that can be controled by the `master`. In the example above, it has a name “test01”. One can start multiple client programs with different names. Then `readdat` is a program to monitor the shared memory data generated by the `client` program.

There are only two messages implemented in this example.

```
master>start test01 1
master>stop test01
```

By starting “test01”, there will be updated information seen by the “reader”.

3 Mechanism of information exchange

3.1 Technical terms

Before going into explanation, some technical words, for which the definition may be different from the usual use, are defined here:

host A network device that has a capability to run the NSM programs. One computer/processor may have multiple hosts as it may be connected to multiple networks.

network A segment of a TCP/IP local area network to which many hosts are connected.

process A process of Unix terminology. There will be no threads or multi-process program used by NSM. NSM in a multi-thread/multi-process program may require a special attention.

node An NSM process that runs persistently and communicates with the other nodes.

daemon A node/process that runs in the background, to handle the details of the NSM communication. A special daemon to maintain the NSM network is called `nsmd2`.

client A node/process that is created by a user using `nsmlib` and communicate with other client nodes. Also called as a user node.

master This word is used in twofold way. A master `nsmd2` is one of the `nsmd2` processes which is the master of the entire NSM network. A `MASTER` node is a predefined client name which may be used for centralized controlling of other clients.

message A unit of data that is used for the communication between nodes.

user A person who writes or runs a client node/process.

3.2 Daemon and client

The NSM network is maintained by two type of programs, daemon and client. A daemon is a unique program that is running on a processor and which should never terminate. A client is a user program that is hooked up to the daemon, started and terminated at any time. A daemon process communicates with multiple clients running on the local processor and daemons running on the other processors. A client process communicates only with the locally running daemon process. This structure is illustrated in Fig. 1.

A client node has a unique node name on an NSM network. Daemons maintain the location of the node and forwards the node to node messages. Therefore, it is possible for a client node to communicate with any other node on a different host, like in Fig. 2-a. If the logical communication structure is close to the case in Fig. 2-b, it is better to run the master `nsmd2` where the requests and messages concentrate in order to reduce unnecessary traffic. In the Belle II tailored library, such a node is predefined and is called `MASTER`.

Communication between a client node and a daemon is based on a TCP connection, a local shared memory and the `SIGRT` signal on a Linux system or `SIGUSR1` on other systems, as shown in Fig. 3. The signal is used to implement the callback mechanism. Note that if you need to run clients under more than one users, `nsmd2` must be running as `root` to allow sending signals (and group id as the same as users as mentioned later). User programs should not be running sa `root` for security.

The daemons are communicating each other, every time when a client node is started or terminated, when a shared memory portion is allocated, freed, opened or closed. One of the daemons (usually the first one) is the master daemon, which maintains the lists of these items. All the lists are maintained on the shared memory and immediately updated on the other daemons. When the master daemon terminates, the deputy (usually the second one) becomes the master and another deputy is selected. All the `nsmd2` daemons make TCP connections to the master and deputy nodes as shown in Fig. 4.

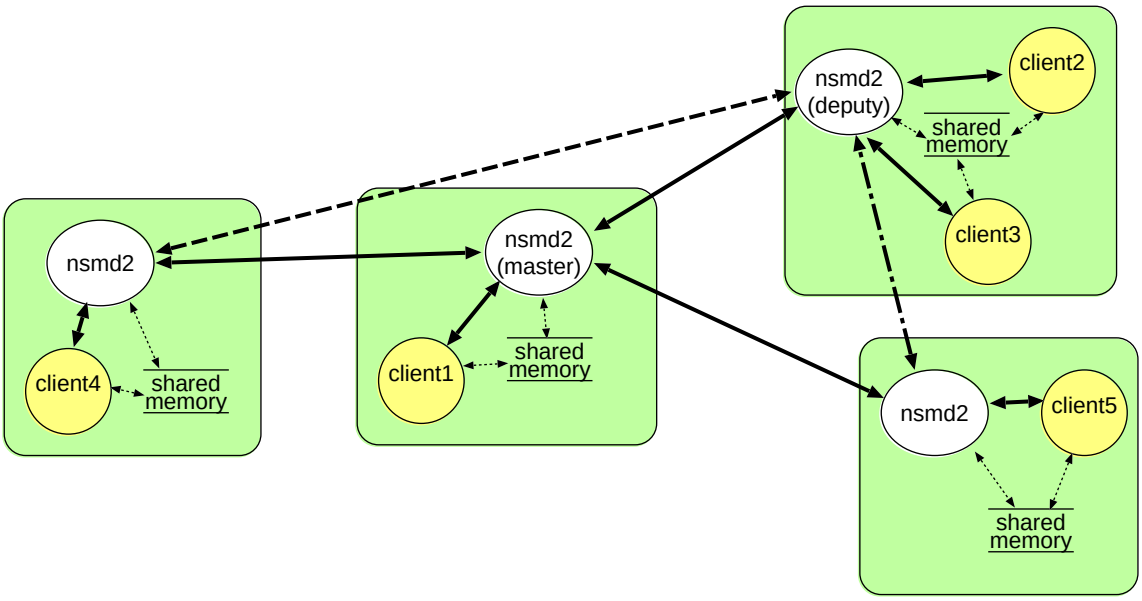

Figure 1: Network structure of NSM. Shaded boxes represent hosts, ovals represent `nsmd2` nodes and circles represent client nodes. Thick Arrows (both solid and dashed) between the nodes and daemons represent TCP connections, and thin dotted arrows represent memory access.

3.3 NSM message

The communication between NSM nodes is made by an NSM message. Each message has a request name, which has to be first registered by the receiving node, together with a callback function that defines the action taken upon receiving the message. One can attach up to 255 words of integers and up to 65535 bytes of byte-streams to a message.

Some of the request names are pre-defined in the Belle II tailored library, and there will be more to be added along with the development of the run control scheme.

These packets are sent as a TCP packet, which is assured to reach the destination in the sending order. However, on a receiver node, a new message may arrive as a new signal while processing the previous message. In some application this is not convinient, so an alternative non-signal based message receiving scheme is now provided by NSM2.

These packets are implemented as a single packet type that contains following fields:

- request id (16-bit integer)
- sequence number (16-bit integer)
- node id (16-bit integer, source node for receiving / destination for sending)
- number of parameters (8-bit unsigned integer)
- byte-stream length (16-bit unsigned integer)
- data buffer for parameters and byte-stream

Conversion between the request name and node name to the request id and node id are taken care by the library, and users usually do not need to aware of them.

The sequence number is usually an increasing integer number to keep the record of the packet ordering, and meant for the trouble-shooting purpose. So it is not meant to be used by users.

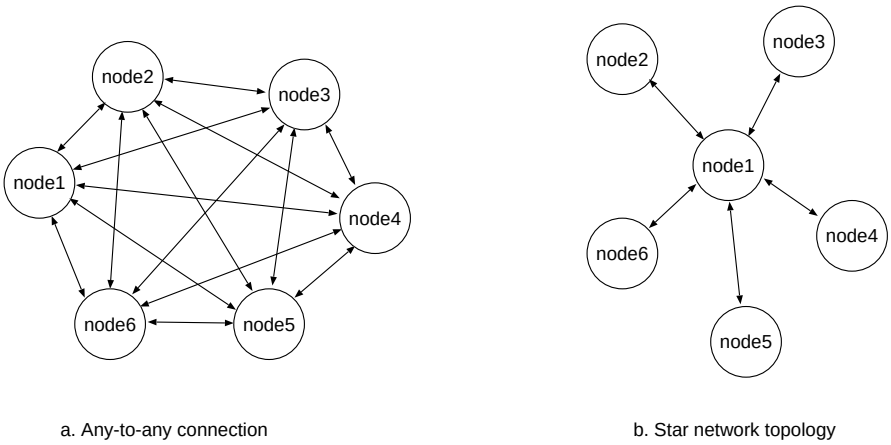

Figure 2: Two types of possible logical structure of the NSM node network.

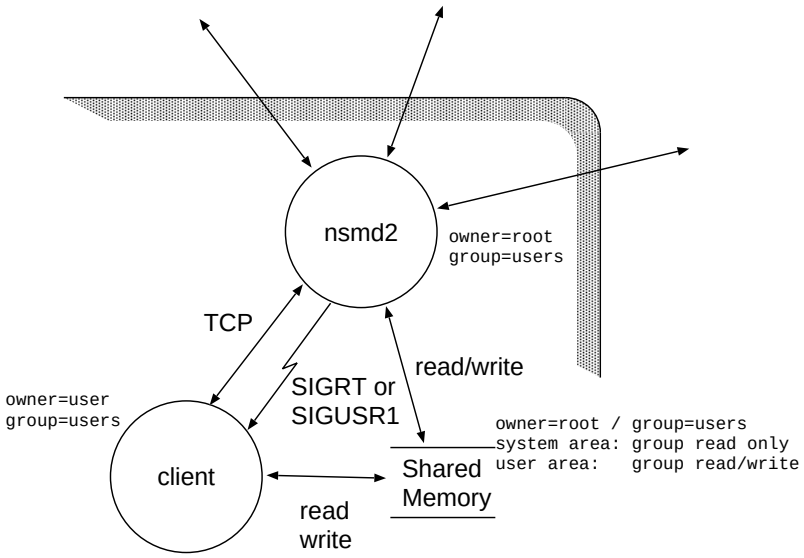

Figure 3: Communication between a client and a daemon.

3.4 Distributed shared memory

The NSM distributed shared memory is implemented using the standard POSIX shared memory system calls for sharing within a host. Therefore, the shared memory is updated immediately between the nodes on the same host. For the inter-host sharing, `nsmd2` distributes the allocated memory area with the UDP broadcast at a given time period (or when requested, this is feature is yet to be implemented in NSM2). The total shared memory size is something that has to be predefined, and it is now 4 Mbyte in the default configuration, which is 16 times larger than that for NSM1.

Since NSM uses the UDP broadcast mechanism, the entire NSM system does not work over two different networks. It does not work either when the netmask or broadcast parameters are incorrectly configured on a host (this could cause a nasty problem to the NSM network). The connection requests or packets from other networks are checked and rejected.

The master `nsmd2` maintains the structure of the shared memory, and a user node must allocate the memory area before using it. Then the memory area is owned by the node until it is freed by the owner node. A unique data name and a data format must be given to allocate a memory area, with which other nodes can access. To access the other's memory area, a user node must open it. The opened memory must be closed

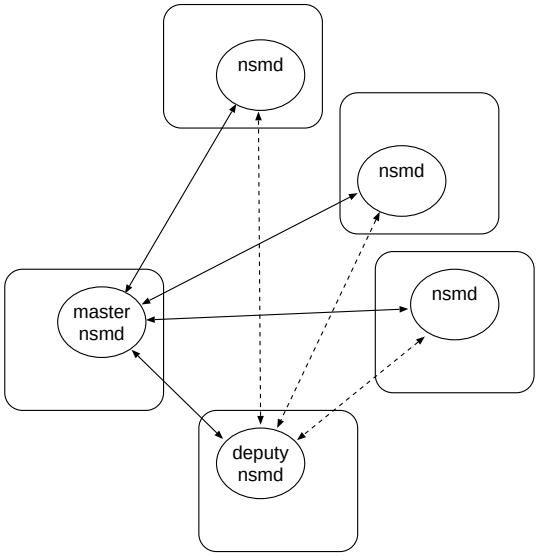

Figure 4: TCP connection between `nsmd2`'s, to master and deputy `nsmd2`.

when it is not needed any more. When a node is terminated, the allocated/opened memory is freed/closed automatically. If the memory is freed (maybe by the termination of the node) but some nodes have not closed it yet, the memory area is not really removed and can be re-allocated by the same node again or opened by the others nodes. The memory area is freed when it is freed and closed by everybody. An example of the memory sharing is illustrated in Fig. 5.

Format of the user data-type has to be known by the NSM daemons, to properly handle the byte-ordering which may be different between different hosts. To easily access the user data and to inform the daemon, an include file has to be written with a subset of the C language for each user data type. This include file defines the revision number of the data type and one struct type.

This include file is included in the user program, and it is also read by the include file parser of the NSM library at run time. The include file parser does not fully support the C language syntax, so the user data type has to be very simple. Since the data type has to consist of data of known size, one has to use int16, int32, or int64 instead of short, int, or long long. The data type should not have a hidden gap, which may be generated by a compiler if an n-byte variable is not aligned at a multilple of n-byte boundary.

Since the size of a UDP broadcast packet is up to 1500 bytes (maximum transmission unit, or MTU) of which a few bytes are used for the header, a data-type with the size up to 1484 byte is updated in a single action. It means a data-type above this size may time-to-time partially updated on other hosts. The maximum size of the user data type is 65296 bytes, which is 44 times the 1484 byte chunk.

The user shared memory area is not really protected for overwriting, but even if a process overwrite a memory area owned by somebody else on a different processor, it does not propagate to other processors and it is eventually filled back by its true owner. However, the memory area of the other local node may be polluted. The system shared memory area is safely protected if `nsmd2` is running as root and clients are running as others. Note the group id of `nsmd2` must be the same as that of clients to allow shared memory reading and writing.

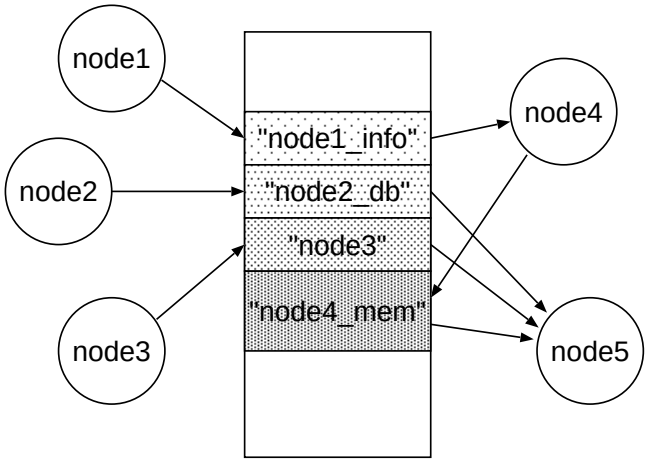

Figure 5: An example of the NSM memory sharing.

4 Running Daemon and Client

4.1 Installation

Installation procedure is not settled yet on this release of NSM2, and therefore no description is given here.

4.2 Running `nsmd2`

When you run `nsmd2`, it starts up as a (foreground) process. This is suitable during the development stage.

The `nsmd2` program cannot start when the default network port or the default shared memory is already in use. When there are two network addresses, it determines the IP address from the default host name. The default port number is 8120 and the default shared memory keys are 8120, 8121.[1]

To change the default behavior, `nsmd2` has following startup options:

`-b`	background mode, log messages are written into a log file.
`-f`	foreground mode, log messages are written into a log file.
`-o`	foreground mode, log messages are written into stdout (default).
`-d[<num>]`	debug mode. Bit 0–7 of num for different debug level.
`-h<host>`	hostname or IP address.
`-p<num>`	port number.
`-m<num>`	priority of the daemon.
`-s<num>`	shared memory key number. (`num+1` is also used)
`-l<dir>`	log file directory.

and the environment variables:

`NSM2_HOST`	hostname for the IP address.
`NSM2_PORT`	port number.
`NSM2_SHMKEY`	shared memory key number.
`NSMD2_DEBUG`	debug mode.
`NSMD2_LOGDIR`	log file directory.
`NSMD2_PRIORITY`	priority of the daemon.

An NSM daemon becomes the master NSM daemon if it founds to have the highest priority, or if there are more than one daemons with the same priority, if it founds to have the smallest IP address among them. Then, the daemon with the second highest priority or the second smallest IP address is chosen by the master to become the deputy NSM daemon. The default priority is 1. To force a daemon to be a master, one can simply give a higher priority with the startup option or environment variable. With this way the master always stays to be the same host as long as the daemon of the particular host is running, and still this daemon is not a critial point as the NSM network does not break down even if this daemon goes down. One can also try to force a daemon not to become a master by setting the priority to zero. However, apparently this daemon still becomes a master if all other hosts also have priority zero and the IP address is the smallest.

One can safely terminate `nsmd2` just by killing it (control-C or `SIGINT` or `SIGTERM`). It then safely kills all the local client nodes and removes the allocated local shared memory. You should not kill `nsmd2` with `SIGKILL` signal unless you cannot kill it with `SIGINT` or `SIGTERM`. If `nsmd2` is abnormally terminated, local shared memory and client processes may remain in the system. At the next start up time, `nsmd2` tries to clean up the shared memory.

[1] The default port number is changed from NSM1.

4.3 Running clients

You can just run the client programs under `example` directory.

`master` — A sample master program in the master-slave model.

`client` — A sample client program that work as a slave of `master`.

`readdat` — A sample program that reads the data written by `client`.

`client_wait` — A non-signal version of `client`.

NSM clients also check `NSM2_HOST` and `NSM2_PORT` variables. All the clients are terminated safely by a `SIGINT` or `SIGTERM` signal (or by control-C).

These programs are written in C++, but no particular C++ features are used.