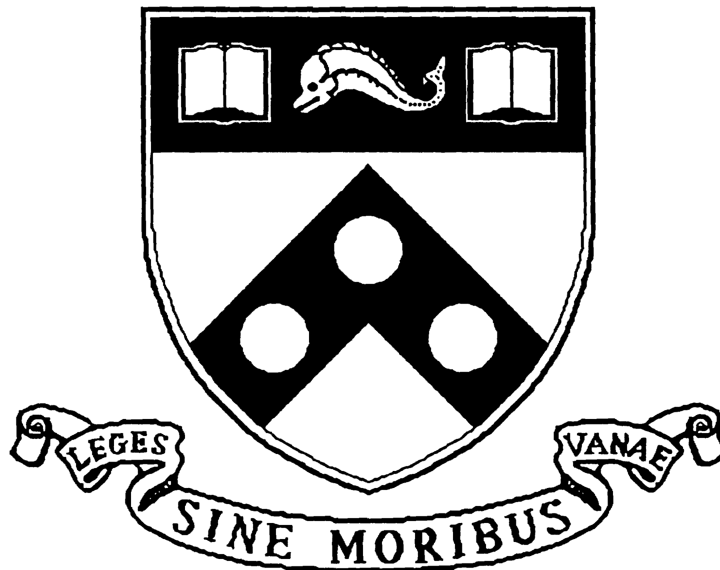
The Mether System: Distributed Shared Memory for SunOS 4.0

MS-CIS-93-24
DISTRIBUTED SYSTEMS LAB 22

Ronald G. Minnich
David J. Farber



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

February 1993

The Mether System: Distributed Shared Memory for SunOS 4.0

Ronald G. Minnich
*Supercomputing Research Center**
Bowie, Maryland
and
David J. Farber
Department of Computing and Information Sciences
University of Pennsylvania
Philadelphia, Pennsylvania

Abstract

Mether is a Distributed Shared Memory (DSM) that runs on Sun[1] workstations under the SunOS 4.0 operating system. User programs access the Mether address space in a way indistinguishable from other memory. Mether was inspired by the MemNet DSM, but unlike MemNet Mether consists of software communicating over a conventional Ethernet. The kernel part of Mether actually does no data transmission over the network. Data transmission is accomplished by a user-level server. The kernel driver has no preference for a server, and indeed does not know that servers exist. The kernel driver has been made very safe, and in fact *panic* is not in its dictionary.

The Mether system supports a distributed shared memory. It is distributed in the sense that the pages of memory are not all at one workstation, but rather move around the network in a demand-paged fashion. It is shared in the sense that processes through the network share read, write, and execute access. And it is memory in the sense that user programs access the data in a way indistinguishable from other memory. The memory is never paged to disk, but the delay of accessing a page over the network is approximately the same as a paging disk.

Two examples of Mether programs are shown in Figures 1 and 2. Note that, aside from the call to `methersetup` these programs look quite ordinary. One program prints out the value of the first 278 bytes of Mether memory; the other clears the first page of the Mether memory and then increments each byte 128 times. If the first program is running the values displayed increase. You can run either program on any host that supports Mether. The writer takes about 8 seconds to run, whether the watcher is running or not. In fact the writer usually runs a little faster if the watcher is on another machine.

As the examples show, programs that access this memory can pretend that it is normal memory. If they do they may pay a substantial performance penalty. As shown in [4] programs that use DSM without modification rarely show the sort of performance gain found on a conventional shared-memory multiprocessor. Programs must be more careful; if they are then they can communicate across the network at apparent memory speeds.

The memory is accessed by opening a special file. Once the file is opened the user program executes an *mmap* system call and maps the area into its address space. From that point on the process may treat the memory as it would any other memory. A function library is provided to make the use of Mether totally transparent.

---

*This work was done while the author was at University of Delaware, Newark, De.

[1] Sun and SunOS are trademarks of Sun Microsystems Inc.

```
#include "world.h"
main()
{
  unsigned int i, j;
  initscr();
  methersetup();
  while(1)
    {
      move(0,0);
      for(i = 0; i < 0x120; i += 0x10)
        {
          printw("%08x:", METHERBASE+i);
          for(j = i; j < i + 0x10; j++)
            printw("%x ", * (unsigned char *) (METHERBASE + j));
          printw("\n");
        }
      refresh();
      sleep(1);
    }
}
```

Figure 1: The Mether watch program.

```
#include "world.h"
main()
{
  int i;
  unsigned char *p = (unsigned char *) METHERBASE;
  methersetup();
  for(i = 0; i < 8192; i++)
    *p++ = 0;
  for(p = (unsigned char *) METHERBASE; *p < 0x80;
      p = (unsigned char *) METHERBASE)
    {
      for(i = 0; i < 8192; i++, p++)
        *p += 1;
    }
}
```

Figure 2: A program that writes to an Mether page

If the process is the only one using an area of the memory, then it will run at full memory speed. If other processes on the same processor are using the same area, they will all run at full speed, unless one of the other processes locks an area of the shared memory. If processes on other processors simply read the memory infrequently there will be a small impact on writes as messages are sent out to the other processors invalidating their copy (or, in the current protocol, updating their copy). If many processors write the same location frequently then there will be a substantial performance degradation, probably only allowing a few thousand operations per second. Mether is non-blocking so the processor will not be slowed down, just the processes accessing the contended-for location.

Mether is inspired by a high-speed memory-mapped network built at the University of Delaware by Delp and Farber. We give a cursory description of MemNet below; for more details see [1], [2] and [3].

MemNet is a memory-mapped network. MemNet provides the user with (in the current implementation) a two Mb contiguous region of memory which is shared between a set of processors. The sharing is accomplished using dedicated page-managment hardware communicating via a high-speed token ring. When a MemNet page is needed and it is not present in the local interface a message is sent over the token ring requesting the page. The hardware provides consistency between pages. The algorithm used is similar to those used for snooping caches: when a chunk is written all other copies of that chunk are invalidated before the write completes. For performance reasons the pages are only 32 bytes long. This size was decided upon as the optimal tradeoff between transmission time and several other factors. For a complete performance analysis, see [1].

On a system such as MemNet the global address space is much larger than any single interface's memory. A problem that must be addressed is what to do in the event a chunk can not find an interface with room for it. Some interface must always keep the space open for that particular chunk (address) in the MemNet address space. To address this problem MemNet supports the notion of *reserved memory*. Reserved memory is the set of chunks for which a particular interface is responsible. Space will always be available for these chunks in the interfaces' reserved area. If no space can be found for a chunk on any interface in a non-reserved area, the chunk will end up back in the reserved memory in the interface which is its home. If MemNet did not support reserved memory, chunks might be lost as interfaces filled up with multiple copies of chunks. In general a MemNet interface will have a "fair share" (i.e. on a system with 10 interfaces, 10%) of its memory as reserved memory, with the rest of the memory available for other chunks.

Mether supports reserved memory too, on a page basis. In fact, a page must be in the reserved memory of some Mether interface for it to be created. In other words, pages are created only from the reserved space, and only when they are referenced. When a non-reserved page is referenced for the first time on a processor, a request for that page is sent out. Only if that page is in some processor's reserved address space will space for it be allocated.

One difference between MemNet and Mether is that Mether blocks the process when a page is unavailable whereas MemNet blocks the processor. This difference is more important than might at first seem. On MemNet, hot spots can consume the process, the network, and all the processors on the network. It is essential that algorithms be well-behaved. Otherwise the processors on the network can, in the absolute worst case, run orders of magnitude slower than normal. On Mether only the processes requesting the information are affected. Other processes, processors, and the network operate normally.

We wanted to gain experience with a DSM that ran on more than the three processors available on the existing MemNet network. Our goal is to build a DSM that matches MemNets' best-case and worst-case performance. In the best case, MemNet runs at memory speeds; in the worst case, it is several orders of magnitude slower. One reason that Mether makes no attempt to minimize paging latency is that we want to get as close to the MemNet environment as possible and explore ways in which to use that environment correctly.

We will describe Mether in further detail below, after which we will describe factors that constrained the design. Mether is driven by MemNet-inspired constraints; there were a number of

other constraints, driven by both technical and political realities.

1 Description

Mether provides an 4 Mb address space which is accessed via the mmap(2) system call. A typical application will open a special device using the Mether library. The *methersetup* function in the library opens the raw Mether device and performs an mmap for the entire Mether address space. As a result of this mmap the kernel does all the housekeeping necessary to support a segment comprising some or all of the Mether address space. Note that on SunOS 4.0 the first mmap is for housekeeping only, and the real mmap does not occur until the process accesses the memory. Therefore we can afford to do the initial mmap; it consumes no Mether resources.

When a page is not present on the local machine the user-level server will request it from the machine currently owning it. Currently the user level server does not support shared read-only pages that span machine boundaries. This enhancement is being implemented.

Once the page is mapped in it may be mapped out again for a number of reasons. To keep a page from being mapped out the process may lock it. While a page is locked no other process may access it. A process may choose to block while waiting for a page to be locked, or it may issue a non-blocking lock request. A blocking lock request will proceed only if the page is present on the local machine and is not locked by someone else. Otherwise the process is blocked until the page is available. Lock requests for a page not present on the local machine will cause a request to be sent out on the network for the page.

The page may also be paged out (i.e. not available on the local machine, and assumed to be present on some other machine). If the page is paged out the mmap call will sleep until the page becomes available.

Up to this point we have discussed Mether mostly in terms of the user's environment. The user calls methersetup and from then on sees ordinary memory, whether the user's processes span machine boundaries or not. In implementation Mether is divided into a kernel driver and a user program. The kernel driver manages the in-memory pages, and the user program manages the transport of pages over the network. There is nothing special about the user program that distinguishes it to the kernel driver. We describe the kernel driver and the user program below.

1.1 Kernel Driver

The kernel driver is responsible for maintaining a set of pages and their associated state. Control of the kernel driver is accomplished using the *ioctl* system call. A table of the ioctl calls and their function is shown in Figure 3.

Initially, only one type of ioctl is supported, the one which initializes the driver. Currently the size of the Mether address space is fixed at 8 Mbytes, and the virtual address range starts at `0x400000`. The parameters which must be set are the base and the size of the reserved address range for the driver. This ioctl is called METHER_INIT. The parameters passed are the start and end of the reserved address space.

Once the driver is initialized, other requests are honored. Until it is initialized all other requests return EINPROGRESS.

There are two ways to lock a page. One is METHER_LOCK, which is a blocking request for a lock. If the page is unavailable due to being locked or paged out the process sleeps. The other way to lock is
METHER_LOCKNOBLOCK, which will return either the error EAGAIN or no error, but which will not block. For both of these if the page is paged out it is marked as wanted. The user level server will attempt to fetch "wanted" pages from other processors.

Name (METHER_)	Parameter	Function
INIT	mether_init *	Initialize the Mether driver. Set the Reserved range.
LOCK	u_long *	Lock a page. The key the driver uses is not the PID, but the minor device number, for reasons explained in the text. If the page is locked or is paged out, the wanted bit is set in the page's status structure. The process sleeps until the page is available. All locks are removed when a process exits if it is that last process holding the minor device open.
LOCKNOBLOCK	u_long *	Lock a page. The same semantics as METHER_LOCK, but if the page can not be locked the driver returns EAGAIN (try again). The wanted bit will be set anyway. This can be used to implement pre-fetching of need pages.
UNLOCK	u_long *	Unlock the page. All sleepers on this address are awakened.
PAGEOUT	u_long *	The page is marked paged out. Paged-out status is not cleared when the marker exits. METHER_LOCK requests and mmap request will sleep on paged out pages, and in addition the page will be marked wanted.
PAGEIN	u_long *	The paged out bit is cleared. The wanted bit is cleared as well. The page is locked; the process doing the pagein must unlock it to make it available to other processes.
FREEALL	u_long *	All minor devices are summarily closed (including the user-level server!) and all pages are freed. The driver will once again honor only METHER_INIT requests.

Figure 3: The *ioctl* calls for the Mether device

The driver keeps track of who locked the page. The identifier used is not the traditional process id but instead the minor device number. This is done so that parents and children can share a minor device number and hence locks. This is not a sophisticated mechanism but has the virtue of being simple, fast, and taking advantage of the way child processes inherit files.

One rule guiding the use of DSM is that if you need a page when you ask for it it is already too late; you are probably going to suffer network latency. Some form of pre-fetching is desirable, and in fact was considered for MemNet hardware. Users of Mether can accomplish pre-fetching in two ways:

- perform the METHER_LOCKNOBLOCK ioctl. If the page is on some other processor it will be marked "wanted" and the user server will fetch it. This is a polling sort of pre-fetch. The process repeatedly performs the LOCKNOBLOCK until it succeeds. Until it succeeds the process may do other work.
- Spawn a child, and have the child perform METHER_LOCK. The child can signal the parent when the page is present. This pre-fetch may be more efficient since there is no polling and no chance of missing the page between polls. It does require more programming at the user level, however.

To unlock a page one uses the METHER_UNLOCK ioctl. All locks are cleared when the open count of the minor device goes to zero (i.e. all potential users of the lock release it). Note that multiple processes accessing a lock need to be careful in this environment because processes that lock a page and exit may leave it locked if child or parent processes still have the minor device open.

A more permanent way of making a page inaccessible is to page it out. A page marked "page out" is not available locally but present on some other machine. Pages acquire the paged out status in one of two ways:

- When the driver is initialized, all non-reserved pages are marked paged out.
- Any user-level program may mark a page as paged-out. Typically only the user-level server does this.

In a sense paged-out is a misnomer; the process marking the page can continue to access it. Indeed, it must if it is to copy it out and send it over the network. The name is an indication of the proper use of the ioctl, however: it should only be used by processes intending to send the page over the network. Mmap calls will block on a paged-out page, as will METHER_LOCK; these calls will also mark the page as wanted.

To page in a page the process uses the METHER_PAGEIN ioctl. Paging in a page will cause it to be locked; the data may be copied in and the page unlocked for use by other processes.

1.1.1 Using Mmap

To access the Mether address space under SunOS 4.0 a segment must be built. The SunOS 4.0 memory management scheme differs radically from other Unix[2] systems in that it is composed of paged segments. Each segment has an address range and fault handlers. For the Mether device the fault handler[3] will invoke the Mether mmap function to get a kernel page frame number. The upshot of all this is that the Mether mmap function gets called at least two times for each page: once when the segment is being built, to see when an address is valid; and once for each time the page is referenced and is not valid. The first "probe" mmap is distinguished by all the protection bits being turned on. Data for a page is never allocated until the first non-probe mmap for that page. If the page is locked by another process or paged out, the mmap will block. Once the page is available the driver makes sure that a physical page has been allocated for use and returns.

[2]Unix is a trademark of AT&T Bell Labs

[3]For those of you familiar with 4.0 Mether works as a segdev

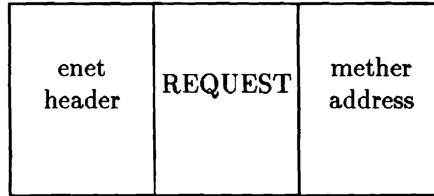


Figure 4: The request packet format

The page may be mapped out of a process's address space at any time, unless the process has locked it. Most often a page will be mapped out when the user level server needs to send it out on the network. Less often some process on the same processor must lock it, usually to prevent it from being paged out. Too much locking is anti-social and may indicate a badly designed algorithm.

When a process locks a page the kernel driver must unmap the page from all other processes that have it mapped in. The Mether device finds the Mether segment in a processes address space and unloads [4] the hardware translation entry for that page.

1.1.2 Using Select

The Mether kernel driver supports the *select(2)* system call. The interpretation is somewhat non-standard. A select which returns "write" status means that a page has just been unlocked by someone, so that the user-level server can take it if it is needed somewhere else. A select which returns "read" status means that someone wants a page. The determination of which page is wanted or has been freed is made by looking through the kernel data structures defining the pages. There are several pages of dynamically allocated kernel data structures which are accessible via the mmap system call. Because of the way the structures change state, it is safe for the user level process to examine and trust the state information found therein. Any change to the state is accomplished via ioctl system calls as described above.

We next discuss the user level server and the protocol it uses.

1.2 User Level Server

The user-level server runs as an event-driven loop. Events are IP-level User Datagram Protocol (UDP) messages and mether page-wanted/page-freed events detected via a *select(2)* system call, or a 100-millisecond timeout on the call. Every 100 ms. the server scans its internal lists of page state and examines the kernel driver Mether page descriptors.

While communications between the current set of user-level servers is via UDP there is no fundamental reason that UDP be the only protocol used.

There are currently three Mether packet types.

The first packet type is a request. If the user-level server sees that a page is wanted, it issues the packet shown in Figure 4. The packet's data consists of nothing more than an address in the Mether address space. There will be one or more packets containing the page returned in response to this packet. On the Sun, for example, eight packets must be returned. Since the UDP available on most BSD Unix systems will not accept an 8 Kbyte UDP message, the user-level servers must do fragmentation and re-assembly.

[4] The function used is hat_unload.

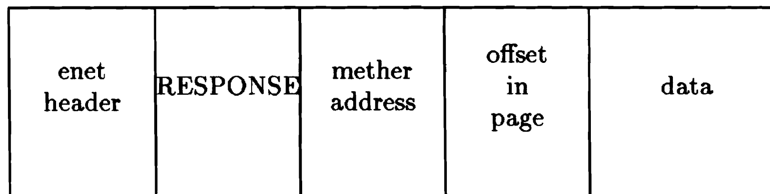


Figure 5: The response packet

There are two versions of the request packet. The first uses the return address of the requester. In this way only the requester need process the packet.

The second packet type, show in Figure 5, is a response. It is about 1 Kb long. It has a return address and a copy of a portion of the page. On a Sun system there are eight return packets for each request. We are currently evaluating the effectiveness of having responses always go to the broadcast address and having a 'watcher' keep track of the state of all the pages.

If the reliability of a TCP connection is desired then a different user-level server can be used, such as the one described in [5].

Note that none of the messages are acknowledged. There is a reason for the lack of acknowledgment which characterizes all Mether operations. On the original MemNet there is no acknowledgement either. There is a significant performance gain to be realized by taking advantage of the low error rate of Ethernet networks. Rather than using traditional error checking mechanisms as are found in TCP/IP, MemNet and Mether depend on the forward error correcting mechanisms supported by the hardware.

2 Design Issues

The design of Mether was driven by both the technical realities of implementing DSM in a system that was never designed for it (Unix), and the political realities of adding an experimental system to workstations being used by researchers for day-to-day computing activities such as mail, text editing, and program development. We needed to use a large number of machines, and such a large number is available to us only on production networks. We discuss both the technical and political considerations below.

2.1 Technical Issues

The first issue to be dealt with was the one which most impacted the design. The question of which operating system to use had been pre-determined: we knew that we would be using some variant of BSD Unix, either the SunOS, Ultrix[5], or Mt. Xinu[6] flavor. The question that drove all other design decisions was how much to modify the kernel. Were we willing to completely redo the paging functions, thus providing a completely transparent paging system? Could we get away with not changing the kernel at all, and allow the user program to trap faults and drive the paging via mmap(3)?

The trade-offs were explored, and in the end we decided not to modify the kernel fault handling code. The reasons were both technical and political. There are many and varied subtleties in the fault handling code in the kernel.

[5]Ultrix is a trademark of Digital Equipment Corporation
[6]Mt. Xinu is a trademark of Mt. Xinu Inc.

It was at that point that we discovered the SunOS 4.0 memory model. The SunOS 4.0 memory model is one of paged segments. Each segment has its own fault handlers. For mmap devices, the fault handler calls the device's mmap function when the user accesses a page for the first time. The implication was that fault handling was done for us. We got all the benefits of modifying, e.g. 4.3BSD, with none of the disadvantages. At this point we restricted our scope to SunOS 4.0 based machines, which was an acceptable decision as they far outnumber any other machine at our site.

The single hardest problem in using SunOS 4.0 is that very little information is available on how to use the new memory model. The new model in our opinion represents a fundamental improvement, comparable to the addition of paging to the Unix kernel. The only way we were able to determine how to use it was to buy sources and hunt through them. We hope to see this situation change.

2.2 Political Realities

It may seem strange to have a discussion on politics in a technical paper, but the fact is that in today's workstation environment politics have an unavoidable impact on a technical design. The reason is simple: in most people's minds, the network is *not* the computer. The computer is the workstation on their desk, and most people think of it as a stand-alone VAX[7] with a thin link to some nebulous collection of other computers. Never mind that their disk blocks and many other resources are provided by the network; if you are contemplating doing anything that impacts their machine they are likely to be quite unhappy. A resource such as Mether may improve the global environment for all users, but typically people only consider the machine on their desk.

As a result the Mether design conforms to several political necessities.

- The kernel driver must be very small and simple. There is no possible justification for increasing someone's kernel size by, say, 100 Kbytes.
- Any kernel structures must also be small and if possible dynamically allocated so that they do not consume space unless the driver is allocated. We use an array of page-descriptor structures each of which has a pointer to a page.
- The driver must never, ever take the kernel down. We have seen device drivers that panic on encountering the most basic inconsistencies. Panicing is unacceptable.
- Any user must be able to shut the driver down at any time. All resources allocated to the driver must be freed.

Having to allow politics to affect a technical design is distasteful. Unfortunately it is a necessary part of any system which wishes to use many workstations.

3 Summary

Mether is an implementation of DSM over Ethernet. The system runs on Sun workstations. The overhead for access to a page not on the local processor is about equivalent to a page fault. The system has recently come into general use, and performance results are very favorable.

Many of the interconnection networks being built for large numbers of processors exhibit the variable latency exhibited by Mether. Learning how to structure algorithms on such a network is an interesting research area and one which we expect to use Mether to explore.

[7] VAX is a trademark of Digital Equipment Corporation

3.1 Acknowledgements

Ron Minnich wishes to thank Maya Gokhale for reviewing early drafts of this paper; Leroy Fundingsland of SRC for the technical discussions of the shared memory implementation on Ultrix; and Gary Delp and Dave Farber for getting him started in this area.

References

[1] Gary S. Delp, Adarshpal S. Sethi, and David J. Farber. *An Analysis of Memnet – a memory mapped local area network.* Technical Report, University of Delaware, Department of Computer and Information Sciences, 1986.

[2] Gary S. Delp, Adarshpal S. Sethi, and David J. Farber. *An Analysis of Memnet: An Experiment in High-Speed Memory-Mapped Local Network Interfaces.* Udel-EE Technical Report 87-04-2, Department of Electrical Engineering, University of Delaware, Newark, Delaware 19716, April 1987.

[3] David Farber and Gary Delp. All systems in sync. *Unix Review*, 7(2):72–77, February 1989.

[4] A. Forin, J. Barrera, and R. Sanzi. The shared memory server. In *Usenix- Winter 89*, pages pp. 229–243, Usenix, February 1980.

[5] Don Libes. User-level shared variables. In *Usenix- Summer 85*, Usenix, 1985.