# Scalable Networked Information Processing Environment (SNIPE)

Graham E. Fagg [a], Keith Moore [a] and Jack J. Dongarra [a,b]

[a] *Department of Computer Science,*
*University of Tennessee,*
*104 Ayres Hall, Knoxville, Tennessee*
*TN-37996-1301, USA*
*email: snipe@cs.utk.edu*
*WWW: http://www.netlib.org/SNIPE*

[b] *Oak Ridge National Laboratory,*
*Box 2008, Bldg 6012,*
*Oak Ridge, TN 37831-6367, USA*

**Abstract**

SNIPE is a metacomputing system that aims to provide a reliable, secure, fault-tolerant environment for long-term distributed computing applications and data stores across the global Internet. This system combines global naming and replication of both processing and data to support large scale information processing applications leading to better availability and reliability than currently available with typical cluster computing and/or distributed computer environments. To facilitate this the system supports: distributed data collection, distributed computation, distributed control and resource management, distributed output and process migration. The underlying system supports multiple communication paths, media and routing methods to aid performance and robustness across both local and global networks. This paper details the goals, design and an initial implementation of SNIPE, and then demonstrates its usefulness in supporting a middleware project. Initial communications performance is also presented.

*Keywords:* metacomputing, message-passing libraries, distributed applications, MPI, PVM, RCDS, multi-path communications

## 1 Introduction

The beginning of the 21st century will present new challenges for large-scale applications involving communication with, and coordination of, large numbers of geographically dispersed information sources, supplying information to a large number of geographically dispersed consumers. Such applications will require an environment which supports long-term or continuous, reliable and fault-tolerant, highly distributed, heterogeneous, and scalable information processing.

Examples of such applications include:

- Indexing and cataloging the worldwide digital library, which will have hundreds of millions of documents, produced at millions of different locations. The collection can be expected to have a high rate of change, both in the number of new documents issued and in the locations by which the documents are accessed.
- Monitoring of weather and prediction of catastrophic conditions to provide planning and decision support for emergency relief.
- Semi-automated air-traffic control on a significantly larger scale, and with greater reliability, than exists today.
- Large-scale same-day shipping of goods over long distances, with automatic rerouting to adapt to equipment failures or weather problems, and load balancing to accommodate fluctuations in traffic.

The characteristics of such applications include: distributed data collection, distributed computation (often in significant amounts), distributed control, and distributed output. Many of these applications will require high reliability and continuous operation, even though individual nodes or links will fail or otherwise be unavailable. Such applications will be constructed out of a wide variety of computational components (including smart sensors, personal digital assistants, workstations, and supercomputers), and a wide variety of communications media (wire, optical fiber, terrestrial radio, satellite) with varying degrees of link reliability, bandwidth, and message loss. The reliability requirement means that such applications must degrade gracefully rather than fail in the presence of node or link failures, or with insufficient communications bandwidth and high message loss rates. Since some computational resources may not be available on a continuous basis, applications may have to adapt to varying computational power. The potential for hostile attack to such systems requires that they have a high degree of security, both for authentication of data and privacy of sensitive information.

To facilitate construction of such systems, we are developing a new programming environment which integrates computational, data gathering, data storage, resource management, and human-computer interaction into a common framework. The framework provides high availability and reliability through replication of both data and computational resources and by careful resource management. This programming environment, called SNIPE, is based on technology developed for the Resource Cataloging and Distribution System (RCDS)

[21] and initially the network messaging layers and process control from the Parallel Virtual Machine (PVM) project[14].

## 2 SNIPEs origins in RCDS and PVM

The design of SNIPE was greatly infulenced by both the RCDS and PVM projects. Both projects provided a starting point for both a code base and for concepts, functions, facilities *and* short comings that could be learned from.

### *2.1 Resource Cataloging and Distribution System*

The Resource Cataloging and Distribution System (RCDS) is designed to facilitate very scalable and fault-tolerant access to network-accessible resources and Metadata, and to provide end-to-end authenticity and integrity guarantees for those resources and Metadata.

RCDS accomplishes this by replicating the resources and Metadata at a potentially large number of (perhaps geographically dispersed) locations. The set of locations for a resource are maintained in a highly distributed and replicated location registry. Similarly, the Metadata for a resource (a list of attribute "name=value" pairs called assertions) are maintained in a separate distributed and replicated registry, which is indexed by the resource's Uniform Resource Identifier (URI) such as a Uniform Resource Locator (URL) or Uniform Resource Name (URN). The Metadata for a resource is self-defining and can contain elements from arbitrary schema or data models. Subsets of Metadata can also be cryptographically signed, using a variety of algorithms, and the signatures provided to RCDS clients. Authentication of resources is accomplished by the use of cryptographic hash functions (such as MD5 or SHA) which are signed by the providers of the information, and made available to clients along with the resource's other Metadata.

A primary design goal of RCDS was to facilitate world wide web access to very large and geographically distributed populations on the scale of the Internet - potentially millions of users accessing the same resource at the same time. Scalability and fault-tolerance were therefore paramount in RCDS's design. In replicated databases there are inherent tradeoffs between consistency among replicas and resource availability in the presence of node and link failures [23,4,22]. When the semantics of the application permit, higher availability can be obtained by using a consistency model which sacrifices strict atomicity and serializability [16].

With its clean separation between replication of data and Metadata, RCDS provides a substrate upon which to implement consistency models of various strengths, according to the needs of the application.

### *2.2 PVM*

PVM is a library and runtime system that transforms heterogeneous networks of workstations (NOWs) and massively parallel supercomputers (MPPs) into a scalable virtual parallel computer. PVM provides an easy-to-use programming interface for several high level languages. It has facilities for process creation and monitoring, inter-process message passing, multicast message passing, and asynchronous signal delivery. It has a simple facility for global registration of well-known services. It allows tasks to detect and recover from failures of other tasks. It is very portable, having been adapted to a wide variety of architectures and operating systems. PVM has been widely embellished, for example, to add security [6] and management of computational resources [7,9], and SNIPE also borrows technology from these efforts.

The PVM system has proven to be highly useful for supporting large-scale distributed scientific applications. However, PVM has limited flexibility, scalability, fault tolerance, and security compared to what is needed for critical information analysis applications:

- PVM allows practical scalability to tens of hosts. While larger configurations are possible, limitations in PVM's resource management and internal state management tend to make such configurations unreliable and inefficient.
- PVM can tolerate slave failures but not failure of its master host. It also cannot tolerate link failures during host table updates. It can tolerate network partitionings only in the sense that hosts that have been disconnected can rejoin the virtual machine for new computations.
- The PVM resource manager uses centralized decision making. This would be a bottleneck for a very large virtual machine. If the single resource manager fails, the default built-in allocation scheme will make inefficient use of computational resources.

PVM lacks a global name space. Process names are valid only within a single "virtual machine." While it is possible for new PVM processes to join an existing virtual machine, a process can only be a member of one virtual machine at a time. This limits PVM's applicability for data gathering and visualization applications, where the data gathered might need to be supplied to multiple computational processes, or the data presented derived from multiple computational processes, which are not specifically determined and configured in advance. PVM provides insufficient security for large or widely distributed applications. PVM lacks built-in facilities for process migration or checkpointing, though it does have low-level system hooks to support condor-based projects [19] such as Co-Check [14][25]. PVM also lacks facilities for redundant data storage. While it is possible for PVM process to run code that is loaded from other network nodes, PVM provides no protection for its hosts against malicious or errant behavior from downloaded code.

## 3 SNIPE and its components

SNIPE originally used the message passing, task management, and resource management aspects of PVM, together with a modified form of RCDS as a framework for replication of resource registries and globally-accessible state.

During development some of PVMs shortcoming already indicated in previously led to the development of a separate non-PVM based communications sub-library within SNIPE. This library was based initially upon the UDP and TCP Internet protocols. Some of the PVM code base has been maintained, in particular the General Resource Manager (GRM)[9] has be modified to allow for redundant resource management processes. Also some of the PVM daemons task startup and signal handling code has been retained.

These facilities have been used as a substrate onto which support for secure execution, checkpoint/restart, and migration of mobile code have been added. The resulting tools (servers and run time libraries) are intended to facilitate construction of very large decision support networks, combining data gathering, computational, data storage, resource management, and human-interaction nodes into a coherent framework. This system contains seven major components: Metadata servers, file servers, per-host SNIPE daemons, client libraries, resource managers, "playgrounds", and consoles.

### *3.1 Metadata servers*

The RCDS based resource catalog servers (RC/Metadata servers) are used to store and provide access to metadata needed for communication between SNIPE processes. This ability to store data which is commonly held internally by distributed systems, in the SNIPE RC servers has allowed for rapid prototyping and implementation of the SNIPE system. This facility has also proven useful to SNIPE user applications as it allows them to share data without the creation of many temporary small files which are usually required. Automatic time stamping of Metadata by the RC servers also helps temporally dis-joint tasks communication by allowing them to decide for themselves the age and therefore relevance of any Metadata previously stored.

Such Metadata includes:

- information about SNIPE hosts (URI-to-address mappings, host architecture and operating system, network configuration, permissions and public keys),
- location and authentication information (public keys and certificates) for SNIPE processes that require global visibility,
- location information for distributed services which provide service at multiple locations, and
- routing information for multicast groups.

In addition, SNIPE Metadata may contain name-to-address bindings for replicated files, including data files consumed or produced by computational nodes, checkpoint files, and mobile code. Finally, the Metadata can contain signed descriptions of mobile code, allowing playgrounds to verify the codes authenticity and integrity and to identify the resources and access rights needed for that code to operate.

Because RCDS resources are named by URLs or URNs, SNIPE processes and their Metadata are addressable using a widely-deployed global name space. Instead of having isolated virtual machines as in the current PVM environment, any SNIPE process can potentially communicate (subject to access control restrictions) with any other process. Thus data gathering nodes and visualization/control nodes can communicate with a variety of computational tasks, not just those in a particular virtual machine.

### *3.2 File servers*

RCDS file servers will be used to replicate files that are used by SNIPE processes, including data files, mobile code, and checkpoint files, and provide access to these files. Replication daemons on these servers communicate with one another, creating and deleting replicas of files according to local policy, redundancy requirements, and demand. Name-to-location binding for these files is maintained by Metadata servers, which are informed as replicas are created and deleted. Access to the files themselves is provided by ordinary file access protocols such as HTTP, FTP, NFS, or CIFS.

### *3.3 SNIPE daemons*

Each SNIPE daemon mediates the use of resources on its particular host. SNIPE daemons are responsible for authenticating requests, enforcing access restrictions, management of local tasks, delivery of signals to local tasks, monitoring machine load and other local resources, and name-to-address lookup of local tasks. Task management includes starting local tasks when requested, monitoring those tasks for state changes and quota violations, and informing interested parties of changes to the status of those tasks (exit, suspend, checkpoint).

### *3.4 Client Libraries*

The SNIPE client libraries provide interfaces for resource location, communications, authentication, task management, and access to external data stores. Resource location allows the client to obtain information about named resources (hosts, processes and data files) including location, characteristics, public keys, certificates, etc. Communication includes message passing, routing (especially between different types of network media), fragmentation, data conversion (e.g. between different host architectures), and optionally encryption, as well as the ability to use different kinds of media (IP, ATM, Myrinet, etc.) Task management includes the ability to initiate tasks (either directly or via a resource manager) and monitor changes in those tasks.

### *3.5 Resource Managers*

Resource managers are tasked with managing resources and monitoring the state of the resources they manage. Such resources can include hosts, processes, and file servers. A resource manager may manage resources for several hosts at once. For the sake of redundancy, any host may be managed by multiple resource mangers. Resource management functions include allocating resources as needed from those available, attempting to adhere to resource allocation goals, and enforcing restrictions on the use of resources. Depending on configuration, resource management may either be "passive" (allowing a process to reserve resources on a particular host, without actually providing the access to those resources), or "active" (where the resource manager acts as a proxy for the requester, allocating resources on its behalf.) In the latter mode, a resource manager may actually suspend, kill, or (if the code is mobile) migrate processes between hosts. Resource managers are also responsible for clarifying requests for resources, selecting the actual resources in response to a request. Finally, resource managers may also be used to manage RCDS Metadata servers according to demand.

### *3.6 Playgrounds*

A "playground" runs under the supervision of a SNIPE daemon and facilitates the secure execution of mobile code. It is a trusted environment which safely allows the execution of such code within an untrusted environment.

The playground is responsible for downloading the code from a file server, verifying its authenticity and integrity, verifying that the code has the rights needed to access restricted resources, enforcing access restrictions and resource usage quotas, and logging access violations and excess resource use. It also provides a run time environment for the untrusted process which generally allows it access to the functions of the SNIPE client library, but which enforces access restrictions.

While SNIPE playgrounds are capable of supporting native code, we anticipate that most mobile code will be written in a machine-independent language such as Java, Python, or Limbo, or some other language specifically designed to allow controlled execution of untrusted code. Implementations of such languages may also be able to arrange the allocation of program storage, in a way that facilitates checkpointing, restart, and migration. When possible, the playground provides hooks for checkpointing, restart, and process migration for use by resource managers.

### *3.7 Consoles*

A SNIPE console is any SNIPE process which communicates with humans. Communication can be via a character-based or graphical user interface. A SNIPE process can also function as an HTTP server, allowing text and graphical output and forms and mouse-click input from any web browser. A SNIPE-based HTTP server can register a binding between a URN or URL and its current location, allowing a web browser to find it even though it may migrate from one host to another, or if the HTTP server is replicated across multiple hosts. SNIPE will make use of standards for Internet resource registration, as those standards are developed. In the meantime, a proxy server is available which allows any web browser to resolve the URI of any RCDS-registered resource (including SNIPE files and processes).

Just as in PVM, there can be multiple SNIPE consoles for any particular application. However due to the highly distributed nature of SNIPE, and the fact that there is no SNIPE *virtual machine* apart from the entire Internet, there is no way to list all SNIPE processes. The state of each process in a process group is maintained as Metadata associated with that process group, which can be queried by any process with appropriate credentials. Similarly, the SNIPE processes which were initiated by the SNIPE daemon on any particular host are registered in Metadata associated with that host.

## 4 SNIPE security model

Authentication is accomplished using public key cryptography. Each principal's public key is stored as an attribute of that principal's RC Metadata. A signed subset of RC Metadata serves as a key certificate. Before a client will consider a signed statement to be valid, the key certificate must itself be signed by a party whom that client trusts for that particular purpose.

In general, each client or service may determine its own requirements for which parties to trust for which purposes. However, certain trust relationships within SNIPE are very common. Since a resource manager must be trusted by the resources that it manages, it is convenient if the resource manager also serves as a certificate authority for users that have permission to use the managed resources, and realms of hosts from which those users may access the resources.

Before the resource manager will grant access to a resource, it must have two verifiable certificates. One is a signed statement from the user, granting a particular process on a particular host, access to the desired resources. The second is a signed statement from the requesting host indicating that the resources are requested by that process. The first certificate is verified by checking the user's key certificate to see whether it is signed by a party that the resource manager trusts to grant access to the indicated resources; the second certificate is verified by checking the requesting host's key certificate. If both certificates are verifiable, and if the requester has permission to access the requested resources, the resource manager then issues its own signed statement authorizing use of the requested resources by that process, and transmits that statement to the hosts where the resources reside. This processes is designed to obviate the need for exposure of any user's public keys. Ideally, the user exposes his public key only to a single trusted host, which issues limited authorizations for access to specific resources. Similarly, a host's public key is never transmitted to any other host.

For the sake of efficiency some of the verification transactions are optimized. For example, rather than having the resource manager separately sign each resource authorization that is transmitted to a managed resource, the resource manager may instead maintain an authenticated connection with each of its managed resources, which is able to detect connection hijacking, and transmit the resource authorization without signatures. Similarly if a particular client host frequently makes requests of a particular resource manager, the client host can establish a secure connection with the resource manager (on behalf of its user) and avoid the need to separately sign each request.

Plans are to provide privacy using the Transport Layer Security protocol [5] with the slight modification that the TLS certificates may be signed RC Metadata in addition to X.509v3 format.

## 5 Implementation Details

### *5.1 Host Environment*

SNIPE nodes can vary in power from personal digital assistants to supercomputers. The only minimum requirement is an Internet Protocol (IP) implementation, though other protocols can be used – either via a gateway (for non-IP capable hosts), or between IP-capable hosts that also share a faster communications medium. While all SNIPE host environments provide communications and therefore the ability to access and manipulate SNIPE-registered resources, a preemptive multitasking operating system with reasonable security is generally required for implementation of SNIPE file servers, daemons, resource managers, and playgrounds. Due to a lack of computational resources, less powerful host environments may lack some security features and thus be restricted on how they access other SNIPE entities.

### *5.2 RC Data Structures*

The Metadata servers hold shared and private data for each SNIPE component and allows for a very open and flexible system, where little is hidden in internal data structures. This enables rapid prototyping of new components and fast modification of existing ones.

The following section lists the types of Metadata used by the main SNIPE components as an illustration of the types of Metadata that can be stored within SNIPE.

### *5.2.1 Host Metadata*

A host is a resource on which processes can be spawned. It may have one or more CPUs, and one or more network interfaces. There may be restrictions on the use of certain CPUs and/or interfaces, which are enforced by the host daemon. The actual management of host resources may be performed by the host daemon or one or more broker processes.

So the RCDS Metadata for a host includes:

- The distinguished URL for the host
- The number and types of CPUs available on that host
- The number and types of network interfaces available on the host
- The data formats supported by the host
- The protocols supported by the host daemon
- The URL of the host daemon
- The URLs of any brokers which manage this host's resources
- Authentication credentials – public keys and key certificates to be used to authenticate the host to potential clients.

The network interface Metadata includes such things as protocol (IPv4, IPv6, Myranet, raw ATM), addresses, per-message latency, and bandwidth. For IP networks, the netmask is also included; for non-IP networks, a "net name" (which is shared by all hosts on that network, but otherwise globally unique) is included. This information is used by the message routing library to choose an efficient path to the destination, taking advantage of fast, private, and/or non-IP networks where available. It is also used to determine where to establish multicast routers.

### *5.2.2 File Server Metadata*

A file server is a host which is capable of spawning "file sinks", which accept data from SNIPE processes to be stored in files, and make that data available to other processes. The files thus stored may be replicated to other locations, and made available by multiple protocols such as http and ftp. The RCDS Metadata for a file server includes:

- A URL for that file server
- The protocols via which data are accepted
- The protocols via which data are provided

A single host may provide both computational and file storage services, in which case both kinds of Metadata are used.

#### 5.2.3 Process Metadata

A process runs on one or more hosts and provides computational or other services. The process Metadata allows other processes to monitor it or communicate with it. The process also has a "notify list" of other processes which wish to be notified if a process changes state. This Metadata includes:

- The distinguished URN for that process
- The supervisor LIFN[3] for the process
- The communications addresses for the process (including interface characteristics such as netmask or net name)
- The "notify list" for that process. Each member of the notify list is a URN of another process.

#### 5.2.4 Multicast Group Metadata

A multicast group is a named group of processes, to which one can send a message as if it were a single process. The actual routing of multicast messages is performed by host daemons which elect themselves as multicast routers on a per-group basis. The RCDS Metadata for a multicast group exists to allow other hosts to find the multicast routers for a particular group, and thus join or leave the group. (It should be noted here, that this type of Multicast group is not designed for high performance of closely coupled processes as in MPI for example, but rather for reliable group communication across the Internet. Although a high performance multicast protocol has been tested, see section 6).

Multicast group Metadata includes:

- The name of the multicast group (a URN or URL)
- The URLs of multicast routers for the group
- A "notify list" of processes that wish to be notified if the set of multicast routers changes.

### 5.3 Unicast Message Routing

Unicast message routing is performed using the RCDS Metadata for the destination process, and the RCDS Metadata for the host on which that process currently resides. If the source and destination are on a common private network or common IP subnet, the message is sent using the fastest of those. Otherwise, the message is sent using the host's normal IP routing.

### *5.4 Multicast Message Routing*

Multicast messages are sent to one or more host daemons which are acting as routers for that particular multicast group. Each router is responsible for relaying messages to a subset of the processes in the group, and to other routers which have not received the message. Whenever a process joins a multicast group, its host daemon heuristically determines (based on the presence or absence of other routers in the group, and the networks to which those routers are attached) whether it should become a router for that group.

For the sake of fault-tolerance, each process wishing to participate in a multicast group may register its membership in the group with multiple multicast routers. Each router which adds itself to the group also registers itself with more than half of the other routers for that group, and any message sent to that group is initially sent to more than half of the routers for that group. This is intended to ensure that there is at least one path from the sending process to each recipient process.

### *5.5 Spawning Processes*

A request to spawn a process is made relative to a particular host, or more generally, to a set of resources named by a URL (of which a URL for a host is a special case). The request is accompanied by a specification of the program to be run and the environment which the program requires. For instance, the program may require direct access to a particular network or resources, it may run only on certain CPU types, it may require a certain amount of memory or CPU time or local disk space. If the program must be run on a particular host, that is also part of the environment specification.

If the RC Metadata for a host contains a list of brokers, the request to spawn is sent to one of the brokers for that host. Otherwise, the request is sent to the host daemon. The host daemon may handle the request itself, or refer the request to a broker.

Whichever host daemon or broker actually the process will also create a distinguished URL for the process and associate the per-process RC Metadata with that URL. This makes the new process globally visible so that other processes can find it and communicate with it.

### *5.6 Process Migration*

General process migration is facilitated by the migrating process initiating its own migration. After migration the process updates RC servers with its new location and also informs other SNIPE tasks on its notify list that it has moved. The original process maybe required to act as a relay or redirect for a short period depending on the communication subsystem characteristics of other communicating peers. Any processes that do not notice its migration will be unable to establish communication with the original task and will find its new location (or pending location) via the RC servers. Processes with open communications are guaranteed no loss of data while migration is in progress. Temporary storage of state is provided by the SNIPE file servers.

Some programming environments are designed to make it easy for a process to be migrated from one host to another without explicit code in the program to perform that operation. For such programming environments, the details of process migration may be arranged by the host daemon rather than the process itself.

### *5.7 Replicated Processes*

Several kinds of replicated processes are supported by SNIPE:

- If several computational processes are run concurrently, provided with the same input, and expected to produce the same result, a multicast group can be created to provide input to all of those processes. SNIPE Metadata can then be created for the new pseudo-process, consisting of all of the processes in the group, and with the multicast group listed as the communications URL. All data sent to the pseudo-process will then be transmitted to each member of the group. However, if multiple processes send data to that multicast group, there is no assurance that each of those replicated processes will receive the data in the same order.
- If it is desirable to provide a service at multiple locations, using multiple protocols, or at multiple hosts, a LIFN can be created for that service, and each of the service locations (URLs) associated with that LIFN. Any process attempting to communicate with that service will then see multiple service locations (URLs) from which to choose.

### 5.8 Playgrounds

A process may be executed on a host subject to certain restrictions. The host daemon is responsible for enforcing those restrictions. If the restrictions are severe, the host daemon may execute the process within a playground. A playground is an environment which enforces restrictions that cannot easily be provided via the normal operating system. It may, for instance, limit access to local files, or to the machine's network interfaces, or the amount of cpu time or memory used.

A playground may provide a restricted environment for the execution of "native" programs, or it may provide an environment for the execution of programs believed to be "safe", such as Java bytecode, safe-tcl,etc. In either case the playground is responsible for verifying the authenticity and integrity of the program, and checking the credentials of the process making the request to ensure that the process has the appropriate permissions.

Implementation of playgrounds varies widely from one platform to another, and not all platforms are capable of imposing the restrictions which may be required without modification to the operating system. Native code playgrounds are complex to implement and difficult to verify. A playground's capabilities are therefore advertised as RCDS Metadata, which can be used by a process or resource manager in scheduling mobile code.

### 5.9 File Servers, Sinks, Sources and I/O

SNIPE file servers provide the ability for SNIPE processes to store data in files and retrieve the data from those files, using the normal message passing routines used to send messages between processes.

- A "file sink" process reads SNIPE messages sent to it and stores them into a file.
- A "file source" process reads a file consisting of SNIPE messages and sends them to a SNIPE address.

Opening a file for writing thus consists of spawning a file sink process which will store its output in such a way that it can be accessed by another process via its URN or URL. Opening a file for reading is implementing by spawning a file source process which reads a particular file (named by a URN or URL) and transmits that to a particular SNIPE address.

SNIPE file servers can also be used to access ordinary data files via URLs and LIFNs, and to export data to files which can then be accessed by external programs using common protocols such as HTTP.

## 6 Current Implementation Status and Testbed

As of spring 1998, the SNIPE system consisted of the following components:

- RC/Metadata servers: based on RCDS using SUN RPC with authentication based on MD5 hashed shared secrets.
- SNIPE communications module: which supported a selective re-send UDP protocol as well as TCP/IP and an experimental multicast protocol for ethernet. Performance figures for 100M-bit ethernet and 155 M-bit ATM are given in Figure 1. The module provided system buffering of messages so that migrating or temporarily unavailable tasks did not result in lost messages and/or data. The system also provided the ability to switch routes/interfaces as links failed without user applications intervention (unless it effected a required QoS for example).
- Simple SNIPE daemons that start and monitor tasks and resources. These daemons provide asynchronous messaging and signal delivery.
- SNIPE resource managers that can select resources based on user application requests and system loading.
- SNIPE file servers. Currently no support is offered for automatic duplication of updates when writing to or modifying a duplicated file. Duplicated file reading/access is supported via location of closest resource daemons.

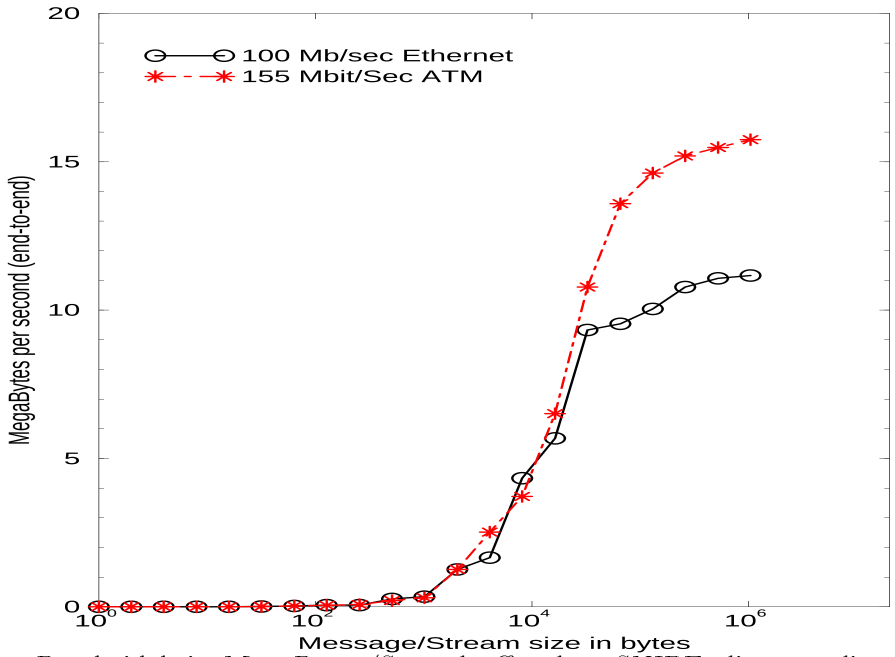



Fig. 1. Bandwidth in MegaBytes/Second offered to SNIPE client applications on various media.

SNIPE testbeds have been running at the University of Tennessee since autumn 1997 and due to replication have maintained an almost perfect level of availability. SNIPE testbeds have also extended to the University of Reading, UK and the Aeronautical Systems Center, Major Shared Resource Center (ASC MSRC), Wright-Patterson AFB, in support of an across MPP inter-MPI application system.

### *6.1 PVMPI/MPI_Connect: A SNIPE Application*

PVMPI[8,10] is a software system from the University of Tennessee that allowed different vendor implementations of MPI-1.1[20] to inter-operate almost transparently. Its original aim was to allow different sub-sections of an application to execute on different MPPs that suited each sub-task and utilized the vendors optimized MPI implementations on each, while still inter-operating across MPPs without having to use a non-optimal implementation such as socket based MPICH-p4.

The *PVMPI* system suffered from the need to provide access to a PVM daemon *pvmd* at all times. On many MPP systems that enforce the use of a batch queuing job control system on top of their native run-time systems as in *PBS-POE* [2] it was not possible to provide con-current access to both a PVM daemon and the MPI application.

Thus PVMPI was modified into *MPI_Connect*, a new system based upon PVMPI that used SNIPE for name resolution and across host communication instead of utilizing PVM. This system proved easier to maintain (no virtual machine to disappear) and also oftered a slightly higher point-to-point communication performance.

## 7 Related Work

Metacomputing frameworks have been popular for nearly a decade, when the advent of high end workstations and ubiquitous networking in the late 80's enabled high performance concurrent computing in networked environments[24,1].

Both Legion[15] and Globe[18] are metacomputing systems based on an Object Oriented view of the world, where processes, files and resources are all considered instances of known object classes. In both systems the *methods* available to access and manipulate objects are rather fixed, although both systems have well designed security and object location services. Their main disadvantages compared to SNIPE are their relative size which can lead to problems of availability and ultimately performance.

Globus[13] is a metacomputing infrastructure toolkit built upon the *Nexus*[12] communication framework. The Globus system is designed around a toolkit that consists of the pre-defined modules pertaining to communication, resource allocation, data, etc. much like components in SNIPE. In fact the communications in SNIPE is much like Nexus and the Metacomputing Directory Service (MDS) has similar aims to the RC servers in SNIPE. A major difference between MDS and SNIPE RC servers is MDS is based on LDAP, a protocol originally designed for use with X.500[17] The RC servers are based on a true master-master update data model and are inherently more scalable.

## 8 Conclusion

SNIPE has provided a useful vehicle for experimenting with a number of issues in current metacomputing research, such as global naming of tasks, files, resources. As well as multiple communication path allocation and management, and a number of different security models.

Although SNIPE is designed as an experimental system to try various research modules on, it in itself has proved useful in producing metacomputing applications and middleware. Its most attractive features are its small size, speed and openness. It enforces no virtual machine, and hides little internal data, and as such allows for very rapid inclusion of new modules, tools and services. When used across the Internet its strong security, scalability and robust communications allow the construction of reliable high quality applications in an unreliable Internet.

## References


[1] M. Baker, G. Fox, and H. Yau. Cluster Computing Review. Northeast Parallel Architectures Center, Syracuse University, November 1995, New York, http://www.npar.syr.edu/techreports/index.html.

[2] A. Bayucan, R.L. Henderson, T. Proett, D. Tweten, and B. Kelly. Portable Batch System: External Reference Specification. Release 1.1.7, NASA Ames Research Center, June 1996.

[3] Shirley Browne, Jack Dongarra, Stan Green, Keith Moore, Theresa Pepin, Tom Rowan, Reed Wade, and Eric Grosse. Location-Independent Naming for Virtual Distributed Software Repositories. University of Tennessee, *Department of Computer Science Tech Report ut-cs-95-278*, February 1995.

[4] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *Computing Surveys*, 17(3):341-370, September 1985.

[5] Tim Dierks, Christopher Allen The TLS Protocol, Version 1.0. Internet-draft "draft-ietf-tls-protocol-05.txt" (work in progress) November 12, 1997. *Eventually to appear as an Internet Request for Comments.*

[6] T. H. Dunigan and N. Venugopal. Secure PVM. *Technical Report ORNL/TM-13203*, Oak Ridge National Laboratory, August 1996.

[7] Graham E. Fagg and Shirley A. Williams. Improved Program Performance using a cluster of Workstations. *Parallel Algorithms and Applications*, Vol 7, pp. 233-236, 1995.

[8] G. E. Fagg and J. Dongarra. PVMPI: An Integration of the PVM and MPI Systems. *Calculateurs Paralleles*, Vol. 8, 1996, pp. 151–166.

[9] Graham E. Fagg, Kevin S. London and Jack J. Dongarra, Taskers and General Resource Manager: PVM supporting DCE Process Management, *Proceeding of the third EuroPVM group meeting*, Munich, Springer Verlag, October 1996.

[10] Graham E. Fagg, Jack J. Dongarra and Al Geist, PVMPI provides Interoperability between MPI Implementations, *Proceedings of Eight SIAM conference on Parallel Processing*, March 1997.

[11] Graham E. Fagg, Keith Moore, Jack J. Dongarra and Al Geist, Scalable Networked Information Processing Environment (SNIPE), *Proceeding of SuperComputing 97*, San Jose, CA., November 1997.

[12] I. Foster, C. Kesselman and S. Tuecke. "The Nexus approach to integrating Multithreading and Communication", *Parallel and Distributed Computing*, Vol. 37, pp 70-82, 1996.

[13] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, Vol. 11(2), summer 1997.

[14] A. Geist, A. Beguelin, J. Dongarra, W. Liang, B. Manchek, and V. Sunderam, *PVM: Parallel Virtual machine - A User's Guide and Tutorial for Network Parallel Computing,* MIT Press, 1994.

[15] A. S. Grimshaw and W. A. Wulf. The Legion vision of a worldwide virtual computer. *CACM*, Vol. 40(1), pp39-45, January 1997.

[16] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Trans. Database Syst.*, 12(2):170-194, June 1987.

[17] S. Heker, J. Reynolds, and C. Weider. Technical overview of directory services using the x.500 protocol. *RFC 1309*, FY14, IETF, 03/12 92.

[18] Philip Homburg, Maarten van Steen, and Andrew S. Tanenbaum. "An Architecture for a Wide Area Distributed System." *In Proceedings of the Seventh ACM SIGOPS European Workshop*, Connemara, Ireland, September 1996.

[19] M.J. Litzkow and M. Livny. Condor–A Hunter of Idle Workstations. *Procs. 8th IEEE Int. Conf. Distr. Computing Systems*, June 1988, pp. 104–111.

[20] Message Passing Interface Forum, *MPI: A message–passing interface standard,* International Journal of Supercomputer Applications, 8(3/4), 1994.

[21] K. Moore, S. Browne, J. Cox and J. Gettler, The Resource Cataloging and Distribution System, *Technical report*, Computer Science Department, University of Tennessee, December 1996.

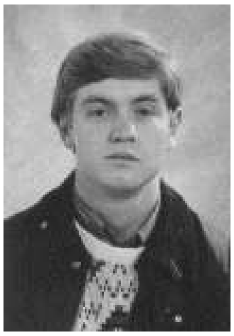
[22] K. Ramarao. Transaction atomicity in the presence of network partitions. *In Proc. Fourth International Conference on Data Engineering*, pp 512-519. IEEE Computer Society Press, February 1988.

[23] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Transactions of Software Engineering*, SE-9(3):219-228, May 1983.

[24] L. Smarr and C.E. Catlett. Metacomputing. *Communications of the ACM*, Vol. 35(6), 1992, pp. 45–52.

[25] Georg Stellner and Jim Pruyne. Resource Management and Checkpointing for PVM, *Proceeding of EuroPVM 95, pp. 130-136, Pub. Hermes, Paris, 1995.*

## Biography of Authors



Graham E. Fagg received his PhD in Computer Science from the University of Reading, England in 1998, his BSc in Computer Science and Cybernetics from the University of Reading, England in 1991. From 1991 to 1993 he worked on CASE tools for inter-connecting array processors and Inmos T-800 transputer systems as part of the ESPRIT Alpha project. From 1994 to the end of 1995 he was a research assistant in the Cluster Computing Laboratory at the University of Reading working on code generation tools for group communications. Since 1996 he has worked as a senior research associate at the University of Tennessee. His current research interests include scheduling, resource management, performance prediction, benchmarking and high speed networking. He is currently involved in the development of a number of different meta-computing systems including: SNIPE, MPI_Connect() and HARNESS.

Keith Moore holds an M.S. in Computer Science specializing in gatewaying between electronic mail systems from the University of Tennessee in Knoxville which he received in 1996. He has worked on PVM, the Heterogeneous Networked Computing Environment (HeNCE), and the Resource Cataloging and Distribution System (RCDS). He currently serves as Area Director for Applications of the Internet Engineering Task Force (IETF). His interests are in networked messaging, computer security methods and scalable internet computing.

Jack Dongarra received the PhD in Applied Mathematics from the University of New Mexico in 1980, MS in Computer Science from the Illinois Institute of Technology in 1973, and BS in Mathematics from Chicago State University in 1972. Dongarra is a Distinguished Scientist specializing in numerical algorithms in linear algebra at the University of Tennessee's Computer Science Department and Oak Ridge National Laboratory's Mathematical Sciences Section. Professional activities include membership in the Society for Industrial and Applied Mathematics and in the Association for Computing Machinery (ACM).