Title

High Performance Computing, High Speed Networks, and Configurable Computing Environments

Authors
Johnston, W.E.
Jacobson, V.L.
Loken, S.C.
et al.

Publication Date
1992

LBL-32161
UC-405

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

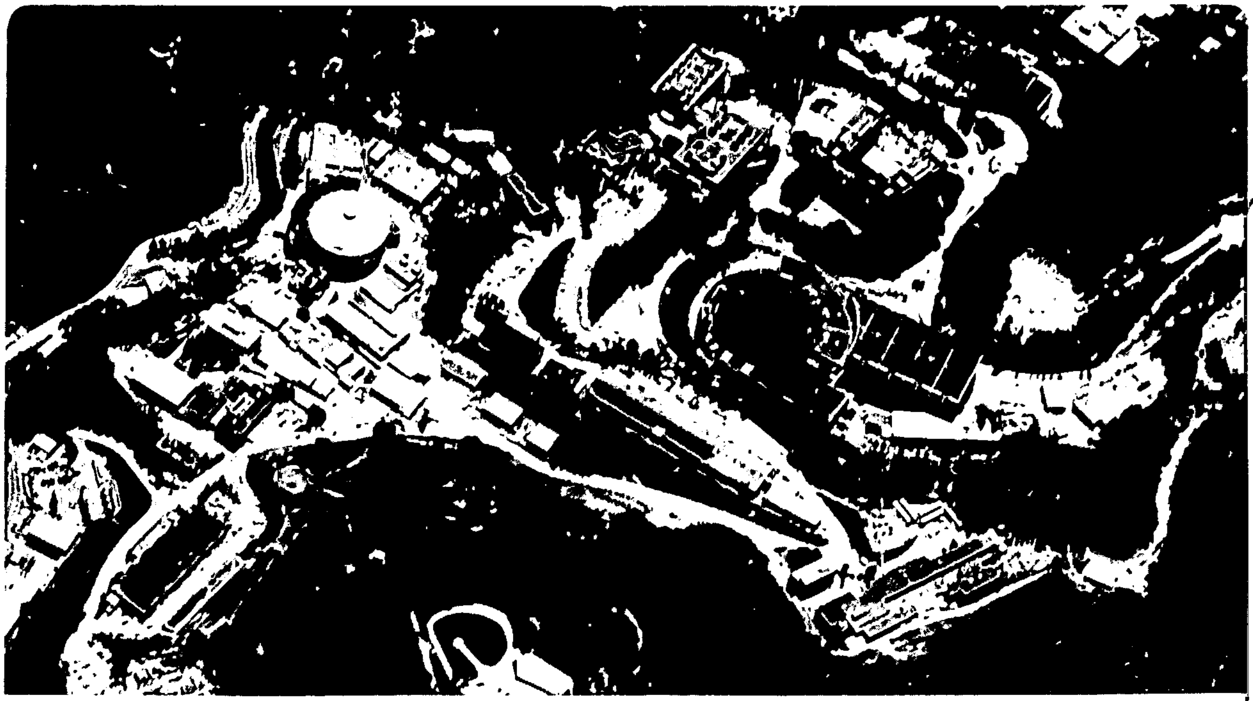
Information and Computing Sciences Division

To be published as a chapter in *High Performance Computing in Biomedical Research*, T.C. Pilkington, editor, CRC Press Inc., Boca Raton, FL, 1992

High Performance Computing, High Speed Networks, and Configurable Computing Environments: Progress Toward Fully Distributed Computing

W.E. Johnston, V.L. Jacobsen, S.C. Loken, D.W. Robertson, and B.L. Tierney

April 1992



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

LBL-32161

High Performance Computing, High Speed Networks, and Configurable Computing Environments:

Progress Toward Fully Distributed Computing

William E. Johnston*, Van L. Jacobson,
Stewart C. Loken, David W. Robertson, and Brian L. Tierney

Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

- Contents -

*For further information please contact johnston@george.lbl.gov (tel: 510-486-5014, fax: 510-486-6363). This work is supported by the U. S. Department of Energy, Office of Energy Research, Scientific Computing Staff, under contract DE-AC03-76SF00098. Any opinions are solely those of the authors and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory, or the U.S. Department of Energy. Trademarks are acknowledged by †. This paper is available from Lawrence Berkeley Laboratory, Berkeley, CA 94720, as report LBL-32161.

Abstract

The next several years will see the maturing of a collection of technologies that will enable fully and transparently distributed computing environments. Networks will be used to configure independent computing, storage, and I/O elements into ''virtual systems'' that are optimal for solving a particular problem. This environment will make the most powerful computing systems those that are logically assembled from network based components, and will also make those systems available to a widespread audience.

Anticipating that the necessary technology and communications infrastructure will be available in the next three to five years, we are developing and demonstrating prototype applications that test and exercise the currently available elements of this configurable computing environment. The Lawrence Berkeley Laboratory (LBL) Information and Computing Sciences and Research Medicine Divisions have collaborated with the Pittsburgh Supercomputer Center (PSC) to demonstrate one distributed application that illuminates the issues and potential of using networks to configure virtual systems. This application allows the interactive visualization of large 3D scalar fields (voxel data sets) by using a network based configuration of heterogeneous supercomputers and workstations. The specific test case is visualization of 3D magnetic resonance imaging (MRI) data. The virtual system architecture consists of a Connection Machine-2† (CM-2) that performs surface reconstruction from the voxel data, a Cray Y-MP† that renders the resulting geometric data into an image, and a workstation that provides the display of the image and the user interface for specifying the parameters for the geometry generation and 3D viewing. These three elements are configured into a virtual system by using several different network technologies.

This paper reviews the current status of the software, hardware, and communications technologies that are needed to enable this configurable environment. These interdependent technologies include: (1) user interface and application program construction methodologies; (2) the interprocess communication (IPC) mechanisms used to connect the software modules of the application; (3) the network protocols and interface hardware used by the IPC for communicating between modules running on separate and independent computing system elements; (4) the telecommunications infrastructure that provides the low level data transfer functions for the networks that connect the geographically distributed elements used by the application; and (5) the nature of the functional elements that will be connected to form virtual systems.

1. Introduction

Advances in software paradigms, computing systems, and communications technology and bandwidth over the next few years will help enable an information analysis environment in which scientists have uniform and unimpeded access to computing and data resources regardless of their geographic location. This environment will provide a "just in time" approach to assembling the resources needed to solve specific instances of problems in computational simulation and data acquisition, analysis, and archiving. It will allow us to design optimal architectures for the solution of specific problems, and then, by using network based resources, to logically assemble and use the required elements only for the time during which they are needed. The environment will also allow us to quickly reconfigure these virtual systems for the purpose of adapting to a changing scientific experiment configuration, or the refinement of computational procedures. The resources will consist of (1) computing elements (workstations, parallel and vector processors, and specialized processors for graphics rendering, compression, and encryption), (2) data handling elements (large, high speed data "buffers", and distributed mass storage systems), data collection/experiment control devices, sensor system input elements, etc., (3) graphics/image display user front end systems, and (4) the software systems to easily interconnect these elements. Not only will the ability to build network based configurations of these resources allow powerful capabilities to be brought to bear on large problems, it will also allow access to these capabilities by a much wider community of people than is presently possible due to the inherently distributed access. No longer, for example, will a very expensive graphics workstation be required to interactively explore large data sets. Instead, a network based configuration of shared computing elements can provide this functionality. A graphics rendering system that is capable of dealing with very large data sets may be built up of network based computational elements, and may be easily shared among collections of researchers, educators, and students who then display the resulting images on low cost workstations located in their own work areas. This environment will qualitatively improve our ability to address the hard and large scale problems typified by the HPCC Grand Challenges [FCCSET], and will provide wider access to large scale computing. This environment will be enabled through advances in several areas:

(1) Improvements in user workstation performance and architecture that will give us an order of magnitude increase in the critical areas of I/O and memory bandwidth, and the routine incorporation of co-processors that will provide practical encryption and video-like image stream compression-decompression.

(2) The national commitment to widely available, high bandwidth networking through the implementation of the NREN[1], the resulting increased involvement of the telecommunications industry, and advances in network protocols and

security architectures will move distributed access to resources, data, and information into the mainstream.

(3) Hardware and software improvements will permit multiple heterogeneous computing systems to be easily configured to routinely cooperate on diverse problems.

(4) Easy access to massive unique data archives will be enabled through advances in data management and mass storage systems, e.g. [Sequoia].

(5) User interface paradigms will evolve to allow non-computer specialists to easily assemble the computing and data handling elements into effective tools to work on scientific problems.

To promote this environment, we are collaborating with several groups to design and implement prototype applications that incorporate several of the critical elements listed above. The purpose of this work is twofold. First, we will make early use of the technology in order to provide concrete experience in identifying and solving various integration problems. Second, we will make the fullest use of current technology to provide useful solutions to problems like the visualization of very large data sets arising from 3D imaging scenarios such as high resolution MRI.

2. The Research Imaging Paradigm

Configurable systems are essential to many scientific endeavors. For example the research imaging environment (illustrated in Figure 1) is characterized by three elements that are typically geographically dispersed. These elements are:

(1) The imaging device and its associated control system;

(2) Computing and data storage elements to provide the processing speed, large memory, high speed data handling, etc., needed to capture and interpret the data;

(3) Workstations for user guidance of the operation of the imaging device, control of the modeling and simulation processes, display of the images resulting from the visualization of the data from imaging device, and for incorporation of remotely located expertise.

The leading edge research in scientific imaging will always involve separation of these components by their very nature. The source of image data is typically a large, unique facility at a scientific/medical research institution. The physical constraints of many advanced imaging systems, such as LBL's Advanced Light Source (a 60 meter

[1] The NREN (National Research and Education Network) is a component of the President's High Performance Computing and Communications initiative.

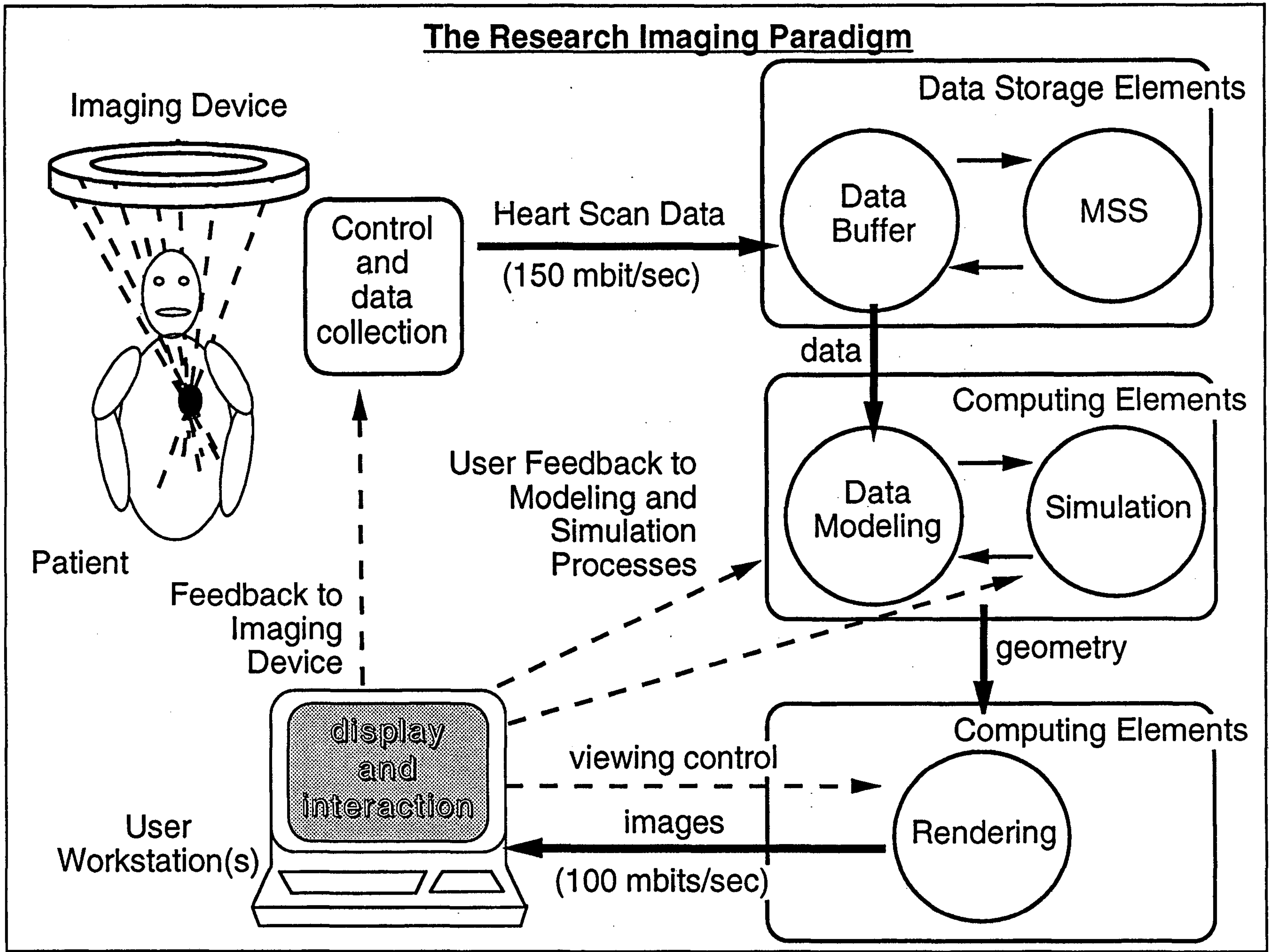

The Research Imaging Paradigm
Imaging Device
Patient
Control and data collection
Heart Scan Data
(150 mbit/sec)
Data Storage Elements
Data Buffer
MSS
data
Computing Elements
Data Modeling
Simulation
User Feedback to Modeling and Simulation Processes
Feedback to Imaging Device
geometry
Computing Elements
display and interaction
viewing control
images
(100 mbits/sec)
Rendering
User Workstation(s)

Figure 1

diameter, 1.5 GeV synchrotron ring producing high brilliance, near-coherent X-rays in a large research complex)[2] or NASA's Hubble telescope, are such that it is not possible to place the imaging system geographically close to the computing resources.

Adequate computing resources are also frequently not available at the imaging device. The data generated for a single ''unit of interest'' (either in time, or space, or both) for this class of imaging devices is large: typically of the order of 1000 MBytes (megabytes). The computation required to convert the data collected by the imaging device to viewable images (e.g. MRI reconstruction), or to do the simulations that are needed to otherwise interpret or utilize the data, require a supercomputer-equivalent capability. The supercomputer is needed for its large memory (gigabytes), or its high processor speed (hundreds of millions of floating point operations per second (megaflops)), and usually both. The experimental nature of the research imaging situation also makes configurable systems advantageous because the algorithms for the data analysis, and therefore the optimal computing environment needed to implement them, may not be well understood. A configurable computing environment is needed during the research stages to allow data sources and experimental, expensive ''super'' computing and high speed data storage elements (which will not be located at the research site) to be flexibly and temporarily combined into optimal systems.

The human participants (the application or experiment scientists, the imaging device design scientists and engineers, and even the operators) may not be conveniently or continuously close to the facility. The research staff must have access to an analysis and visualization workstation in order to interpret the highly complex image data, to guide the operation of the imaging device, and for medical study, diagnosis, and treatment planning during the operation of the imaging device. This workstation must provide interactive visualization of the results of the reconstruction and simulation done on the supercomputer, and provide for interactive feedback to the imaging device, and the object being imaged (a patient in the case of a medical physics facility).

A versatile, easily used, and distributed suite of software is necessary to couple the elements of this environment in order to do the imaging, image reconstruction, image analysis and simulation, and imaging device control; and to allow for algorithm development. This last is a very important step. The imaging devices in question are research facilities, and the data always requires substantial transformation before it is meaningful for human interaction and analysis. The ability to easily and quickly prototype and modify software algorithms for this image reconstruction is crucial.

All of the above components are resources that an ''information age'' environment will have to smoothly integrate, regardless of location, in order to realize the

[2] See, for example: ''The ALS Life Sciences Center'', Lawrence Berkeley Laboratory, PUB-5234, 1989, Berkeley, CA, 94720

potential benefit to a broad based audience.

The challenge facing us over the next few years is to address the key issues, and remove the bottlenecks that inhibit a configurable environment. The current problems for widely distributed elements are: (1) the sources of data are either isolated from the supercomputers, or communicate over low speed networks; (2) the supercomputers are at best connected to the graphics and analysis workstations via low to medium speed networks; (3) adequately fast and economical workstations are just barely available, (4) the current software environment is not well enough integrated to permit the flexible development and experimentation needed for the various software algorithms that are at the heart of the imaging and visualization; and (5) the mechanisms of communications are not fast enough.

To solve these problems we need the following:

(1) Adequate network connectivity and bandwidth between the three geographically dispersed components described above. This entails not only networks and network access techniques, but network management, high speed protocols and routing algorithms, and high speed network gateways.

(2) When the network bandwidth is adequate, the next bottleneck will be the computer/network interface. Interfaces and interprocess communication mechanisms must be improved in order to permit computer systems to utilize high speed data paths without so much overhead that all of the CPU resources are consumed just getting data from the network, into computer memory, to the user process, and back.

(3) Once the data paths operate at high speed, the next problem will be that of providing faster processing through interconnection of various computer architectures that are optimal for various parts of the problem (for example through the use of both coarse and fine grained parallel systems).

(4) Finally, when all of the components are in place we must have a software environment that will permit the integration of these components into a system that will provide the distributed control, data management, modeling and simulation, and visualization and analysis that are needed to solve the problem. [3]

The solution to the first three of the above problems involve eliminating a set of bottlenecks. As each bottleneck is relieved, another will emerge as the problem to solve next. Most of the issues are understood at this point (or at least currently the

[3] See, for example:
"The Software Bus: A Vision for Scientific Software Development", D. Hall, W. Greiman, W. Johnston, A. Merola, S. Loken, *Computer Physics Communication*, 1989.

topic of active research efforts), and the solutions are, for the most part, computer communications or software engineering problems. There are no conceptual barriers to implementing the configurable systems architecture described above. The solution of each of these problems will lead to an immediate increase in the ability of the computing environment to support new scientific and medical physics imaging techniques. The solution of all of these problems will enable new areas of scientific and medical investigation.

3. Anatomy of a Distributed Application

Most of the technologies that will enable the environment described above are involved in building distributed applications. In this section we will describe the technologies that support distributed computing: networking, interprocess communication, and the tools to construct and manage distributed programs. We will also explore how distributed computing is done by describing the architecture of a hypothetical application which is similar to the case study presented in the last part of this paper.

The hypothetical application (see Figure 2) consists of a client process that handles the user interaction and a graphics front end, and two server (worker) processes[4] that operate independently to carry out the computations invoked by user interactions, and then return the results. The exact nature of the application is not significant, but for the purposes of discussion assume that one server operates on a data set to produce a geometric representation, and the other server operates on the geometric data to produce a raster (image) representation.

Many distributed applications are written using the client-server model in which one process (the client) requests service of another process (the server). This relationship implies an asymmetry in the sense that the server responds to client requests, and while this is true for the example (and the case study) it is not terribly significant. The servers can (and do in the case study) communicate among themselves as peers in the course of accomplishing work for the client.

Our exploration of the distributed application will be done by tracing the data flow between the modules (the front end client, server one, and server two). We will briefly describe the software and hardware of services needed to accomplish the data exchange, including the operating system services, and the network and telecommuni-

[4] A process is the basic unit of executing software in a modern computer system. User programs execute as processes, as do many utility functions like print servers, file servers, data base servers, etc. The operating system is the software (not usually considered a process) that manages the physical resources of the computer system and coordinates all of the processes running on the system. Due to software and hardware limitations traditional microprocessor operating systems like DOS and MacOS do not provide for multiple processes in the flexible way that, for example, Unix does.

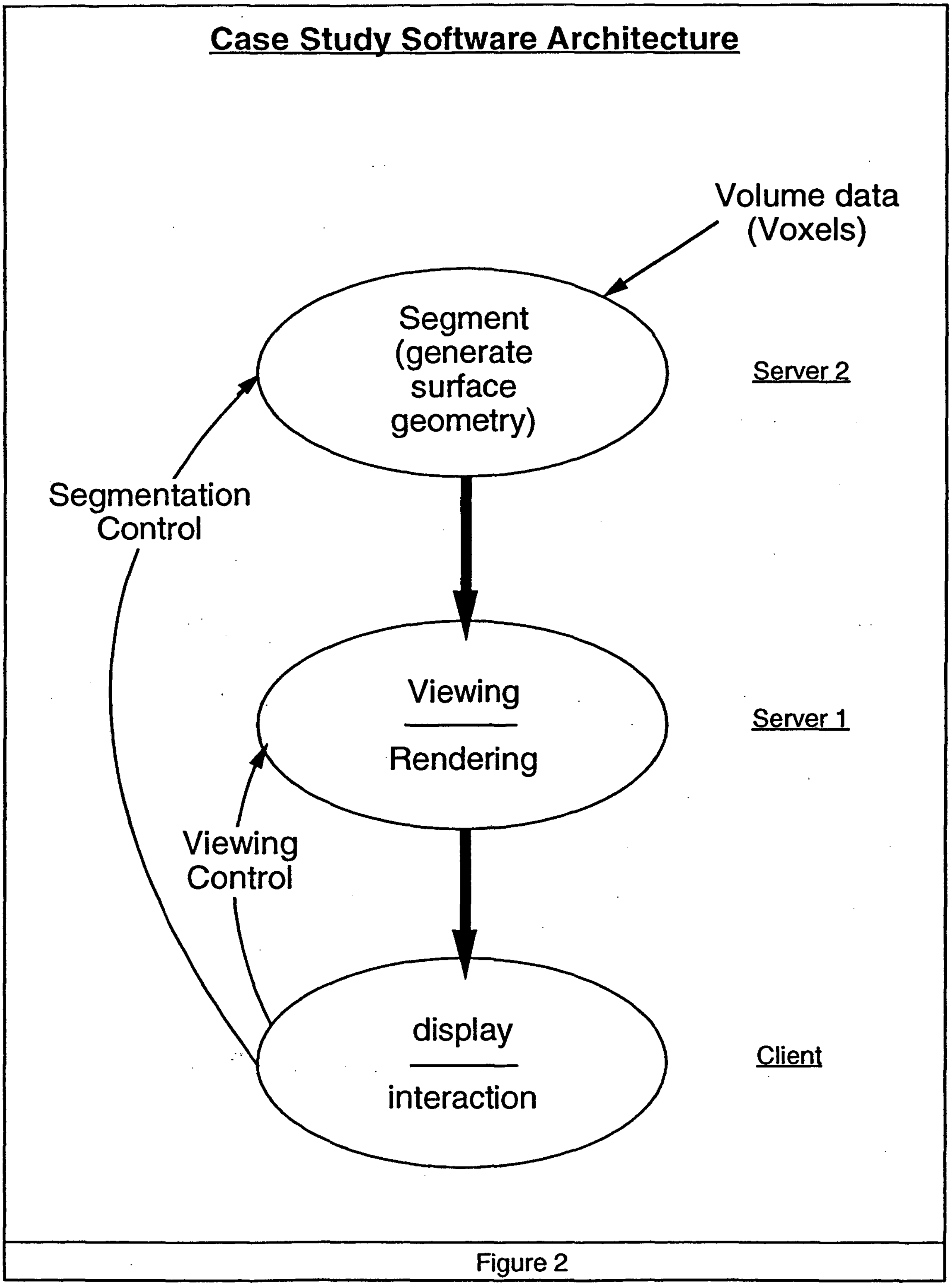

Case Study Software Architecture
Volume data (Voxels)
Segment (generate surface geometry)
Server 2
Segmentation Control
Viewing
Rendering
Server 1
Viewing Control
display
interaction
Client

Figure 2

cation facilities. The descriptions are give in terms of a Unix environment[5], but the principles are broadly based and found in many operating system environments.

3.1. Interprocess Communication Mechanisms

Probably the most important architectural element of distributed computing is the one that provides for processes communicating with each other. There are a number of paradigms in use, and we will describe three of them, but concentrate on one. The first two are *streams* and *messages*, and the third is *remote procedure call.* These all provide interprocess communication (IPC), which involves collecting data in one process, identifying the process that is to receive the data, and managing the actual transfer of the data between the processes. The data movement is done between endpoints of communication and involves moving data into and out of a data marshaling area (buffer) in a process (program) on a particular host (computer system). There may be numerous such endpoints in a process that communicates with many other processes. IPC provides the mechanisms of establishing the endpoints, sending data between them, and managing multiple endpoints.

Streams in Unix are reliable data flows between two endpoints of communication. The data is transferred without structure (e.g. records) or interpretation (i.e. no unit of data - a collection of bits or bytes - has significance in the transfer process) between one process and another. The semantics of the data are entirely up to the processes at each end.

Datagrams are (usually) unreliable messages. They are unreliable in that neither the order of delivery, nor even delivery itself, are guaranteed. Datagrams can be faster than reliable streams in an environment where errors are infrequent (e.g. local area networks); however, the application process has to take the responsibility for error detection and correction when using datagrams, and the mechanisms for this are usually expensive when they actually have to be used. (Datagram error detection does not include worrying about data corruption (bit loss or transformation) because this is detected at a lower level, and results in the packet being discarded.) Like streams, the datagram mechanism does no interpretation or translation of the data.

Remote procedure calls (RPC) are (at least to programmers) perhaps the most intuitive IPC. RPC allows a client process to contact a server process and invoke a procedure (subroutine) on the server for solving a specific task. RPC provides facilities for moving data to and from the server, in much the same way that the functions in a local subroutine library are used. For example, in the same way that procedure calls allow the communication of complex data structures through their argument list, so do RPCs. This capability is typically accomplished by using a data definition

[5] For a beginning user's guide to Unix see [Bims]. A concise engineering overview of the Unix

language that allows one to describe the structure of the data on the client in such a way that it can be serialized for transmission across the network to the server. This is done by defining routines to encode (serialize) and decode the data (recreate the original data structure). For the basic data types (e.g. different types of numeric representations - floating point, integer, bit, etc.) known to the serialization language there is also a mechanism provided for converting from the internal representation of one computer system to that of another. An example of a data definition language is Sun Microsystem's XDR (eXternal Data Representation). (See, for example, the first 20 or so pages of [Corbin].) Some of the issues with RPCs include the overhead of using this data encoding when moving large amounts of data, and the fact that, in following the semantics of conventional procedure calls, RPCs typically block (stop) the client while the server performs its task (though this behavior is not hard to avoid.) The underlying transport for RPC can be either reliable streams or unreliable datagrams.

Interprocess communication is more than transport. It has to deal with all of the issues that are necessary to allow versatile communication between processes. This includes (potentially) resource discovery, server instantiation, establishment of the communication end points and connections, and management of multiple connections to multiple servers.

Resource discovery is necessary to gain access to already existing servers. It can be accomplished by: (1) tables describing well known services (e.g. mail, file transfer (e.g. ftp), remote terminal service (e.g. telnet)); (2) broadcasting a request for service and then using anyone who responds in the affirmative (many Sun RPC based services use a variation of this approach), or; (3) a directory service can be contacted for a list of servers. In all of these cases, the server (or its proxy) is assumed to be running (i.e. available to be contacted).

In the case of a user created server, typically the client process will explicitly start the server. One mechanism commonly used to accomplish this is for the user to have an account on the target host(s) where the server is to execute, and then to use a remote execution service to start the server when it is needed. Let us assume that the compiled server program is in a known location on the remote host, and is executable by the user. On BSD Unix[6] systems the *rsh* program provides the remote execution service. *rsh* starts up a shell (command interpreter) on the remote host, and then uses this shell to start the server which is specified as a command line argument to *rsh*. For an rsh of the form:

operating system may be found in [Sun85].

[6] "Berkeley Standard Distribution" (BSD) Unix is the variant developed at U. C. Berkeley in the 1980s that defined and incorporated the functionality needed for distributed computing. Most of the innovative features of BSD Unix are now provided in all Unix implementations. See [Libes] for a history of Unix.

```
rsh remote_hostname -l username server_name server_options
```

an example of starting a server might look like:

```
rsh example.lbl.gov -l johnston "isosurf -d -port 6001"
```

Once the server is running the next step is to establish communication between the client and the server. When servers are started they typically go into a waiting state for an incoming connection request. Servers may have several communication endpoints corresponding to different sources of data or different functions. Each endpoint for incoming connections is uniquely identified, and the client knows the identities (perhaps because the servers always use the same endpoint identifiers, or because the identifiers are given to the server by the client when the server is started as in the example above). Now that the server is running there are a lot of variations of how it might function. The server has access to all of the resources of the remote machine, and it might, for example, access databases that are already (or permanently) on the remote system. The server might be given data from the client that it then stores on the remote host for the lifetime of the server, and then operate on that data in different ways according to requests from the client. In any event, once the servers are started they wait for the client to send a request to do work. Once that request is received, the servers perform the requested task, and then commence sending the results back to the client. At this point, the outgoing data transfer from the server will typically block until the client initiates the receive. The sending and receiving are done over already established connections, and the blocking is like any other sort of I/O wait.

In our hypothetical example, after passing a request to the server the client does not wait for the results, but rather continues to process local requests and manage the user interface. When the server is ready to start returning the results of a previously initiated request, the client must recognize this and take action to accept the results. As an example of this see Figure 3. Consider that the client is managing the visualization of a large voxel data set. The representation of the data is as a rendered version of the whole data set, and as a low resolution rendering, perhaps surrounded by a wire frame shape to assist in establishing the 3D geometric relationships in the display. The full resolution rendering might take several seconds to generate on the server, but the low resolution version, and the wire frame geometry can be very quickly generated on the client system. In this example, the user requests a change in the view, the client contacts the server to redo the high resolution rendering, and then locally generates and quickly displays the low resolution view. While the server is working on rendering the high resolution version, the client will continue to process user requests. When the server is finished rendering it will send the new image back to the client. In order to get the newly rendered view the client must recognize (1) that the server is trying to contact it, and (2) interrupt what it is doing in order to receive the new image and display it.

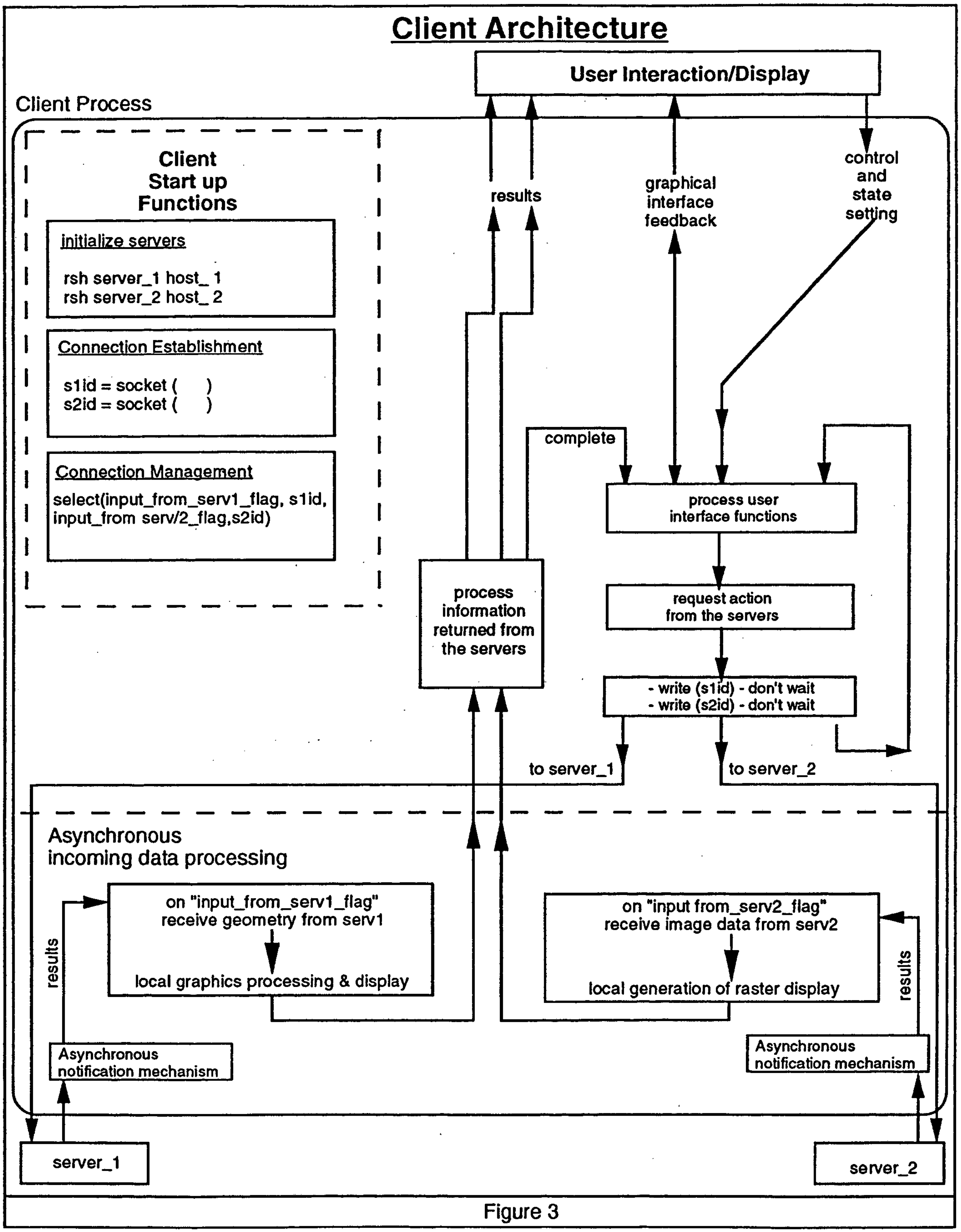

Client Architecture
User Interaction/Display
Client Process
Client Start up Functions
initialize servers
rsh server_1 host_ 1
rsh server_2 host_ 2
Connection Establishment
s1id = socket ( )
s2id = socket ( )
Connection Management
select(input_from_serv1_flag, s1id, input_from serv/2_flag,s2id)
results
graphical interface feedback
control and state setting
complete
process user interface functions
process information returned from the servers
request action from the servers
- write (s1id) - don't wait
- write (s2id) - don't wait
to server_1
to server_2
Asynchronous incoming data processing
on "input_from_serv1_flag" receive geometry from serv1
local graphics processing & display
on "input from_serv2_flag" receive image data from serv2
local generation of raster display
results
results
Asynchronous notification mechanism
Asynchronous notification mechanism
server_1
server_2

Figure 3

This circumstance illustrates the problem of connection management. In our BSD Unix based example, signal-driven I/O is the key to connection management. Basically, there is one signal that notifies the client process that one of the connections is waiting to deliver data from a server. The notification is in the form of an interrupt that causes control to be passed from the current execution thread in the client to an interrupt handler routine. Within the interrupt handler the *select* function can be used to determine which of the several possible connections (corresponding to the several active servers) is waiting to deliver data. Once the connection has been identified, then the client routine can perform the appropriate tasks. The ''asynchronous incoming data processing'' routines (Figure 3) perform the local part of the server specific action, for example receiving an image that was computed on the server and displaying it in a window on the local workstation. When the processing of the incoming data is complete the signal handler passes control back to the client process where it was originally interrupted.

3.2. Networking

Geographically distributed computing is built on networking. The potential today for any reasonably skilled programmer to design and implement a distributed application can be traced directly to the decision of the DoD's Advanced Research Projects Agency (now DARPA) in the late 1970s and early 1980s to fund first ARPANet (the direct ancestor of today's Internet), and then Berkeley Unix (which integrated networking in a fundamental way into the computing scenario). (See, for example, [Stevens]). There is currently an enormous amount of energy going into networking because of its potential for connecting people and distributing information in ways never before possible. The current ''era'' in networking can probably be said to have started with the institutionalization of general purpose scientific and educational networking through the construction of the NSFNet (the backbone, the regional networks, campus connections, etc.), and all that this has led to. The NSFNet has experienced exponential growth for several years, leading to widespread recognition of the potential of computer networking technology. The next paradigm shift will come through the integration of the public telephony network with the computer data networks. This merging is being fostered explicitly through the vision of people like Bob Kahn and Vint Cerf of Corporation for National Research Initiatives (CNRI), Ira Richer, formerly of DARPA, and Steve Wolff and Darlene Fischer of NSF. These people, as well as many others, hypothesized that the way to affordable high speed networking was to get the common carriers (commercial telecommunications providers) involved in computer networking in a much more intimate way than just providing wires (or fiber) as they do now. The current gigabit network testbeds (organized by CNRI, with funding from NSF and DARPA) all involve partnerships with regional or long haul telephone companies. (See [IEEE] for an overview of the testbed activity, also see [Markoff] for a popular exposition of the vision for high speed networking described above.)

The following sections introduce the computer network and telecommunications technologies that are the basis for the networking that we use to build distributed applications.

3.2.1. Models of Networking

There are currently four or five models of networking in common use: IBM's SNA, DEC's DECNet, a collection of PC networks (many based on the Xerox's XNS), Apple's AppleTalk, ISO's X.25 (et al), and the Internet's IP. Arguments can be made that the ISO, Open System Interconnection (OSI) model is acting as a focus for the convergence of these different models (see [Cysper]), but in the scientific and educational communities IP networking is still by far the dominant factor. It is useful, however, to look at the OSI model for networking for several reasons: The terminology is being retrofitted to most of the network models, and it is the networking language of the telephony community.

We briefly discuss the OSI model because is provides a convenient point of reference, but the work in this paper is all based on the IP model of networking. It is worth noting that there are significant differences in approach in OSI and IP networking. (See [Comer], Section 10.6 ''Differences Between X.25 and Internet Layering''.)

The Internet Model of Networking

Briefly stated, the Internet model has all services built on top of unreliable datagrams that are routed through the network on a best effort basis. End-to-end reliable data transfer is the responsibility of the network software in the end hosts. When we say that TCP provides reliable data streams, this is accomplished by the host TCP software, not by any other part of the network. When using TCP, data from the application is fragmented and placed in IP packets. These packets are routed through the Internet by store and forward packet routers. At any point, if for example a router is congested, some of these packets may be lost. Further, there is no guarantee (nor even presumption) that all of the packets that contain the data of the TCP stream will take the same route through the network (which means that the packets may not arrive in the same order in which they were sent). The routers typically connect many different types of link level subnets (the low level networks like Ethernet, FDDI, and point-to-point links that provide the actual data transfer) which implies that the original IP packet may be further fragmented at any point in the network (all of the reassembly is left to the receiving host.) This use of heterogeneous link level networks is called ''internetworking'' (networking among networks).

At the receiver end, the TCP code checks for corrupted data, lost packets, and out of order packets. Packets are reordered, lost or damaged packets are requested to be resent, and eventually a buffer containing information identical to that which was sent is presented to the application for reading. We will talk about the interaction of IPC with network data transport below. We infer that, but say little about the details of how, TCP accomplishes reliable data transfer (see [Comer]for details); and we briefly discuss how packets are transferred (routed) from the sending host, through an internet, to the receiving host in Appendix A.

The OSI Model of Networking

The OSI architectural model of networking is useful both for the standard terminology that it makes available, and because, at the lower layers, it provides a point of commonality with the telecommunications industry. The model is based on functional layering, and the notion (like the Internet model) that there are points in the hierarchy where elements of the information are the same on both sides of a connection. (This is essentially the same as saying that on every participating host there are certain well defined interfaces in the IPC mechanism, independent of any part of the underlying implementation, where everything is constant and well understood.) There are many places in the OSI hierarchy where network services can be obtained. In particular, the notion of reliable data transfer extends all the way down to level two (the subnets). This is different from the Internet model which leaves reliability to the end nodes (under the assumption - so far correct - that more efficient implementation is possible by not replicating services at many layers.) Appendix B contains a brief introduction to the OSI terminology that is used by both the computing and telecommunications industry. In outline form, the OSI model can be represented as:

- Applications and applications support (Level 7)
- Presentation (representation of data in a common format) (Level 6)
- Session (dialogue coordination) (Level 5)
- Transport (data transfer) (Level 4)
- Networking (addressing and routing) (Level 3)
- Data Link (subnet access) (Level 2)
- Physical (Level 1)

Common Carrier Technology

The signal carrying facilities in the Internet have traditionally been the only part of computer networking provided by the common carriers. While this continues to be true, increasingly the common carrier services are moving up the OSI architectural hierarchy as deregulation allows the offering of more value added services. In particular, very high speed networks will almost certainly be provided as broadband

Integrated Services Data Network (B - ISDN) services (mostly through the ATM packet network mechanism described below). This is at least in part due to the increased cooperation between the computing and telecommunications industry, and in part because high speed networks are so expensive that a cooperative effort is required. The services provided by the common carriers are outlined here, and briefly described in Appendix C.

- T - Carrier Systems (OSI Level 1)
  Digital point-to-point links, e.g. T1 (˜1.5 Mbits / sec) and T3 (˜45 Mbits / sec)

- SONET (OSI Level 1)
  Optical fiber based synchronous transmission systems, e.g. OC-3 (155 Mbits / sec), OC-12 (622 Mbits / sec), and OC-48 (2.4 Gbits / sec)

- Frame Relay (OSI Level 2)
  A new service that will provide an upgrade path from X.25

- SMDS (OSI Level 2)
  Switched Multi - megabit Data Service is a metropolitan area network service

- ATM and B - ISDN (OSI Level 2)

Given the probable importance of ATM networks to the future of high speed computer networking, we offer the following comments. Today's telephony networks are built from components that switch and multiplex synchronous streams of information, both analogue and digital. The analogue parts of the network are voice circuits, and the digital parts are the data circuits mentioned above. The circuits can only be switched to paths with similar electrical and bandwidth characteristics, except that circuits can be multiplexed into and out of higher bandwidth circuits.

Computer communications networks, like the Internet, are packet switched networks. On entering the network, the data is broken down into packets that have source and destination addresses, and these packets are sent through dissimilar, asynchronous networks as independent units of data. Since there is no synchronization (each packet is, from the point of view of the network, independent of every other packet) the switches (routers) that move packets around the network can interconnect physically very different types of networks. All that has to be done is that the packet be transformed from the format of one network to another. This is called ''internetworking'' (networking among networks). The problem with computer data networking is that when you need synchronous services (e.g. voice and video), the relatively large packets that traverse heterogeneous networks make synchronization difficult to accomplish.

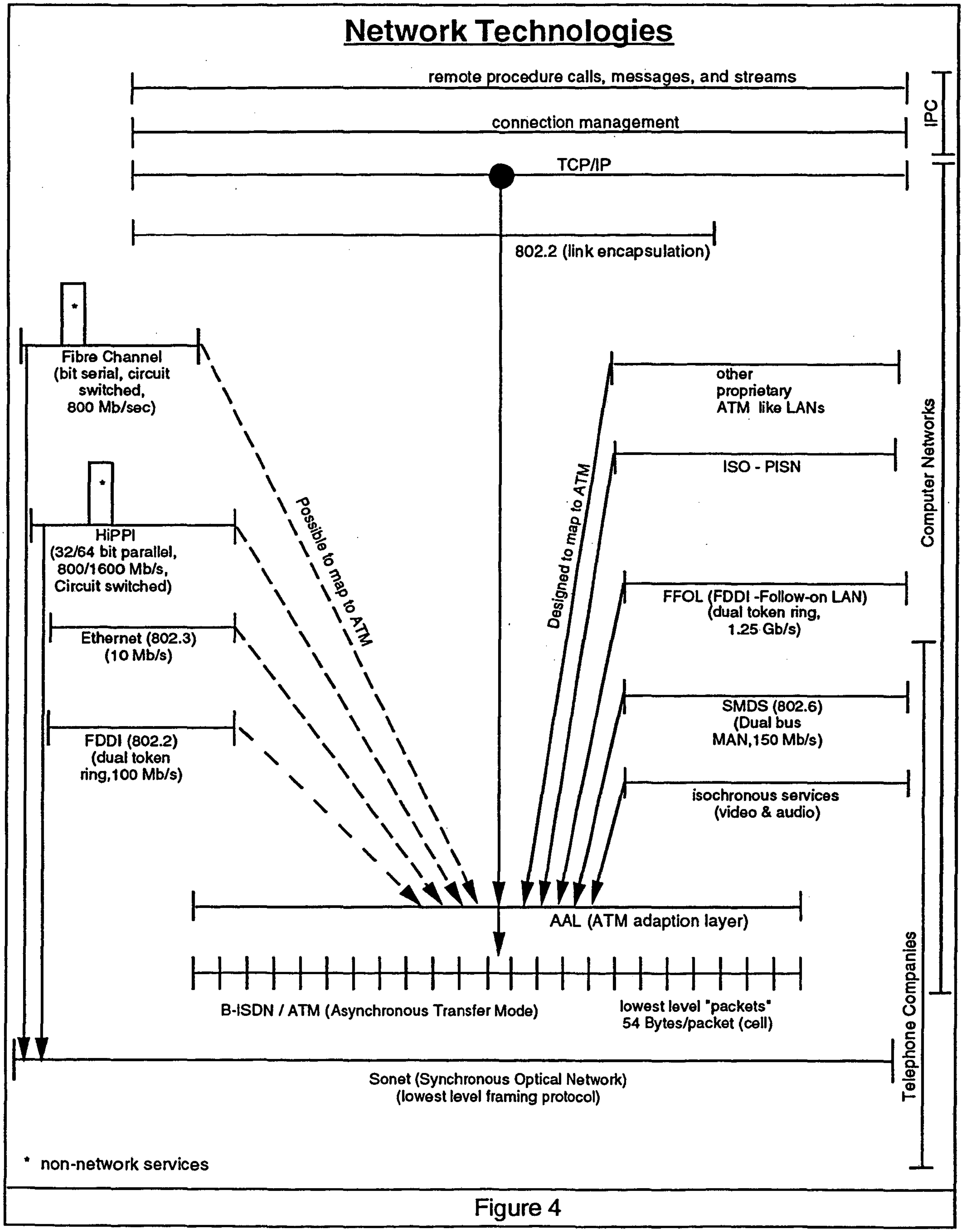
The next generation of telephony networks have been designed to accommodate both the requirements of synchronous and asynchronous services. This work is being done in the international telecommunications standards organization (CCITT), with increasing participation of the international computer standards organization (ISO). One result of this cooperative effort is a network technology that is called Asynchronous Transfer Mode (ATM). ATM packets are very small compared to traditional data network packets (53 bytes as opposed to 500-1000 bytes), and the switches are designed to be very fast. The combination of small and fast is intended to permit synchronous services to be provided over ATM networks. (The Broadband Integrated Data Services Networks (B-ISDN) are a collection of services that will be provided on ATM networks.) ATM networks are also rapidly emerging as the most important high bandwidth computer networks (currently 0.6 to 2.4 gigabits/second is typical). Several joint experiment and development ventures are underway between the computing and communications communities in the national gigabit testbeds, where the ATM technology is being used for experimental high speed computer networks. (See [IEEE]).

3.2.2. Networking Overview

Figure 4 summarizes some of the interrelationships of the technologies described in the previous section. At the top we have the aspects of the technology that are closest to the application: the IPC interface, followed by connection management and the transport protocols. Below transport is the internetworking protocol that deals with routing between (probably heterogeneous) subnets. For the IEEE 802.xx standardized, non-ATM subnets, the 802.2 link encapsulation provides a mechanism for a measure of standardized interface to these link layer protocols, though in many current IP implementations this encapsulation is not used. The adaptation layer of ATM provides assistance for preserving information from the higher levels (for example, the IP packet boundaries) that are needed for efficient segmentation (breaking data units down into packets) and reassembly. In the case of direct use of ATM by IP, ATM is the link layer. Many other protocols are, and will be, mapped to ATM because ATM will be providing the high speed packet (cell) switched data communication mechanism in the future, replacing (in many cases) the current ad hoc collection of subnets (e.g. T-carrier lines). The left side of Figure 4 indicates a collection of currently used link level protocols that can be (and probably will be) mapped to ATM. The right side shows a collection of newer protocols that have been designed to map to ATM. The horizontal bars are intended to indicate (roughly) the use of one layer by another. Figure 5 illustrates how these technologies might look in the (not too far distant) future network.

3.2.3. Networking and Distributed Computing

To see how the networking technologies fit together, it is useful to look at a hypothetical network implementation in the context of the current example. This



Network Technologies

remote procedure calls, messages, and streams
connection management
TCP/IP
802.2 (link encapsulation)
IPC
Computer Networks
Telephone Companies
*
Fibre Channel (bit serial, circuit switched, 800 Mb/sec)
*
HiPPI (32/64 bit parallel, 800/1600 Mb/s, Circuit switched)
Ethernet (802.3) (10 Mb/s)
FDDI (802.2) (dual token ring,100 Mb/s)
Possible to map to ATM
Designed to map to ATM
other proprietary ATM like LANs
ISO - PISN
FFOL (FDDI -Follow-on LAN) (dual token ring, 1.25 Gb/s)
SMDS (802.6) (Dual bus MAN,150 Mb/s)
isochronous services (video & audio)
AAL (ATM adaption layer)
B-ISDN / ATM (Asynchronous Transfer Mode)
lowest level "packets" 54 Bytes/packet (cell)
Sonet (Synchronous Optical Network) (lowest level framing protocol)
* non-network services

Figure 4

network (illustrated as "Customer Site 2", in the lower left corner of Figure 5) is somewhat unusual in that it is an all ATM network; that is the local network is ATM as well as the wide area network. Local ATM networks are different from conventional LANs in that there is a point to point connection from every host to the local ATM switch. Broadcasting on an ATM network will probably also work somewhat differently, with the switch performing this function. We will analyze our hypothetical architecture by following data going out of a server process, through the network, and then back up through the network hardware and software on the client host, to the client process.

The key concepts follow closely with the layering ideas above, and one of the important concepts is that at comparable places in the architecture the format of the data is the same. Let us also elaborate on the concept of an "endpoint" of communication. The fixed part of the network addressing (that is the Internet part which is seen unchanged at both ends of a connection, as opposed to various link layer addresses that come and go as the IP packets make their way through the Internet) consists of the source and destination Internet address, the transport protocol type (e.g. TCP or UDP), and an identifier, called a *port*, that is process specific. In the case of TCP, a connection is identified by the five tuple {transport protocol; source port number; source IP address; destination IP address; destination port number}. The communications endpoint establishment process associates an I/O descriptor with the connection (i.e. an identifier that can be used by the process to send and receive data). It is this I/O descriptor that is used in the connection management in order to tell what connection is waiting for I/O.

Data from the memory of the server process is sent to the TCP connection using, for example, the *write* function (in BSD Unix this is the same write function that is used for file I/O). This stream of data is fragmented into TCP "segments", and the transport type, and source and destination port numbers added. These segments are then (logically) passed to the IP layer, which adds the source and destination Internet address. The IP packet is now passed to the link layer, to go through a process of "adaptation". This involves formatting the IP packet into a "frame", which consists of the IP packet, plus padding to guarantee that the length is a multiple of 48 bytes, and a checksum. An ATM cell (i.e. small packets) header containing the ATM virtual circuit identifier for this connection is constructed, and the frame is then fragmented into ATM cell payloads. The ATM cells (header plus payload) are then sent to the ATM interface on the host, and from there to the first ATM switch. Regardless of how it is determined, the routing in an ATM network is represented as tables in the switches that say where to send the cells for each virtual circuit that the switch knows about.

Unlike most IP packet routers, ATM routers (switches) carry the notion of a

Next Generation High Speed Networks
(Common Carrier Based)

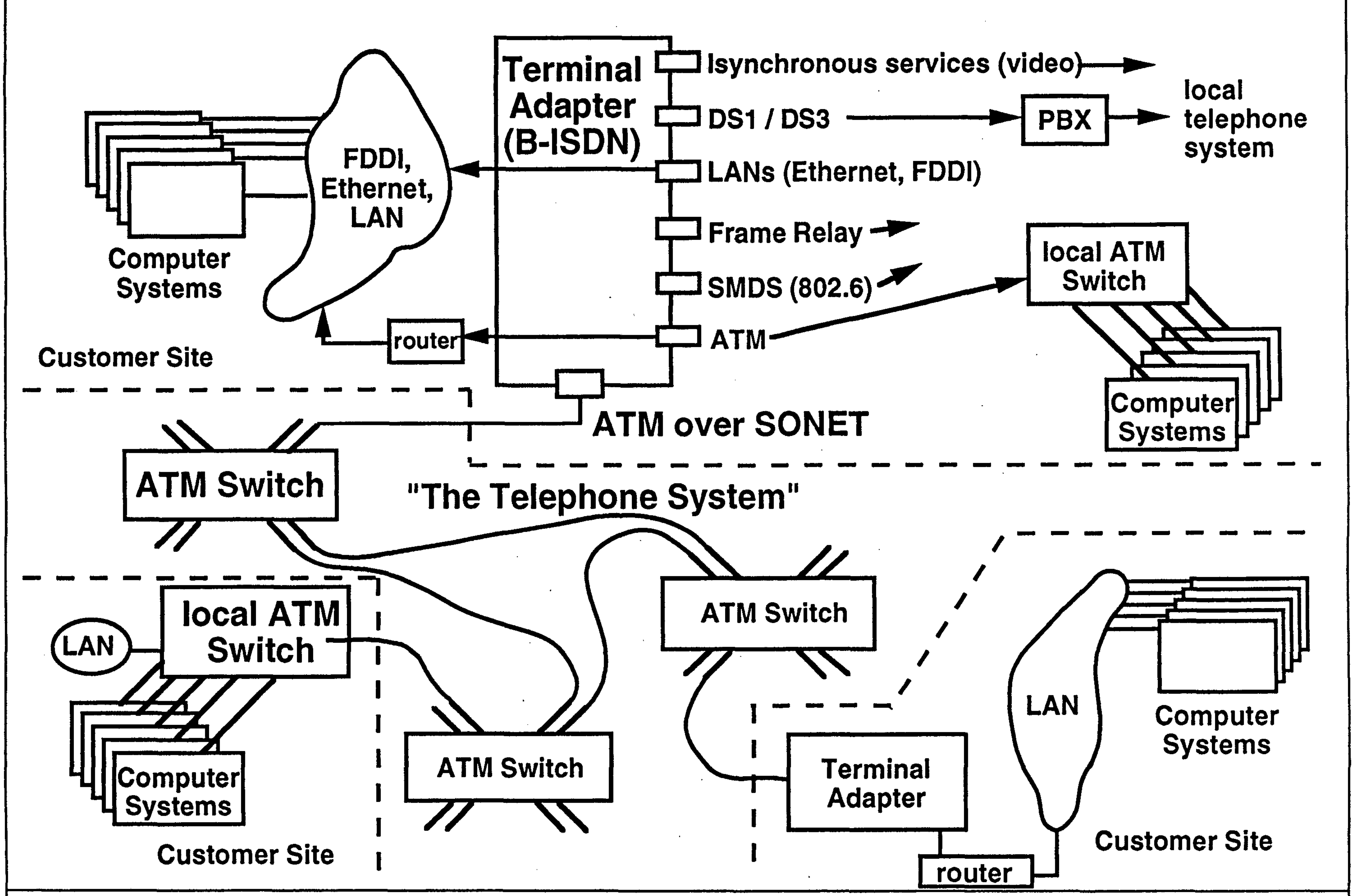


Figure 5

connection all the way through the network. In an IP network, routers do not know of the existence of circuits, but ATM switches deal primarily with circuits. Real circuit switches set up "electronic" circuits to connect two points (that is, the switch sets up a connection between an input port and an output port). The connection is essentially "hard-wired" and the switch just directs a digital data stream from one port to another (this is the case for HiPPI and Fibre Channel switches). A circuit switch does not see packets on the circuit. ATM switches are called virtual circuit switches because while they do switch circuits, they do so by the mechanism of routing packets (cells). A virtual circuit is set up by an entry in a routing table in the switch, and cells are routed by a (relatively small) "route identifier" in the header. The global addressing (e.g. IP addresses or telephone numbers) are dealt with by the software that sets up the circuit (represented as a table entry in the switch.)

At the other end of the network (illustrated in Figure 6), the cells start showing up at the interface of the client system. The progress up through the layers of the receiver can be characterized as a series of de-multiplexing steps. In other words, at each point where a new piece of information about the packet is exposed, it must be acted on. Referring to Figure 6, ATM cells are first separated (1) on the basis of whether they have arrived on a data carrying virtual circuit, or a special virtual circuit that carries control and ATM network information. Next, (2) the frame is reassembled from the ATM cells of the virtual circuit corresponding to our connection. The protocol type of the frame is determined, and the protocol packet is handed to (in our case) the IP network layer (3). We now have recovered the same IP packet that was sent from the other end, and it is identified by the TCP connection "5-tuple". The next layer of de-multiplexing (4) is to determine the transport protocol (TCP in this case). Now the data is extracted from the TCP segment, and placed in a buffer. Since the client is not waiting to read the data, but has enabled an asynchronous notification mechanism (5) for this connection, that mechanism is invoked, and the client process is notified that data is ready to read. When the client gets around to executing a *read* on this connection (6), the data is transferred into the memory of the client process, and the interprocess communication is complete. The client now has available the results of the work of the server.

4. Configurable Systems

We have now examined the IPC and network technology that will allow us to make process-to-process and system-to-system connections. The remaining technologies needed to make configurable systems practical are user level software to specify and use the configured system, and the functional elements themselves.

Interprocess Communication

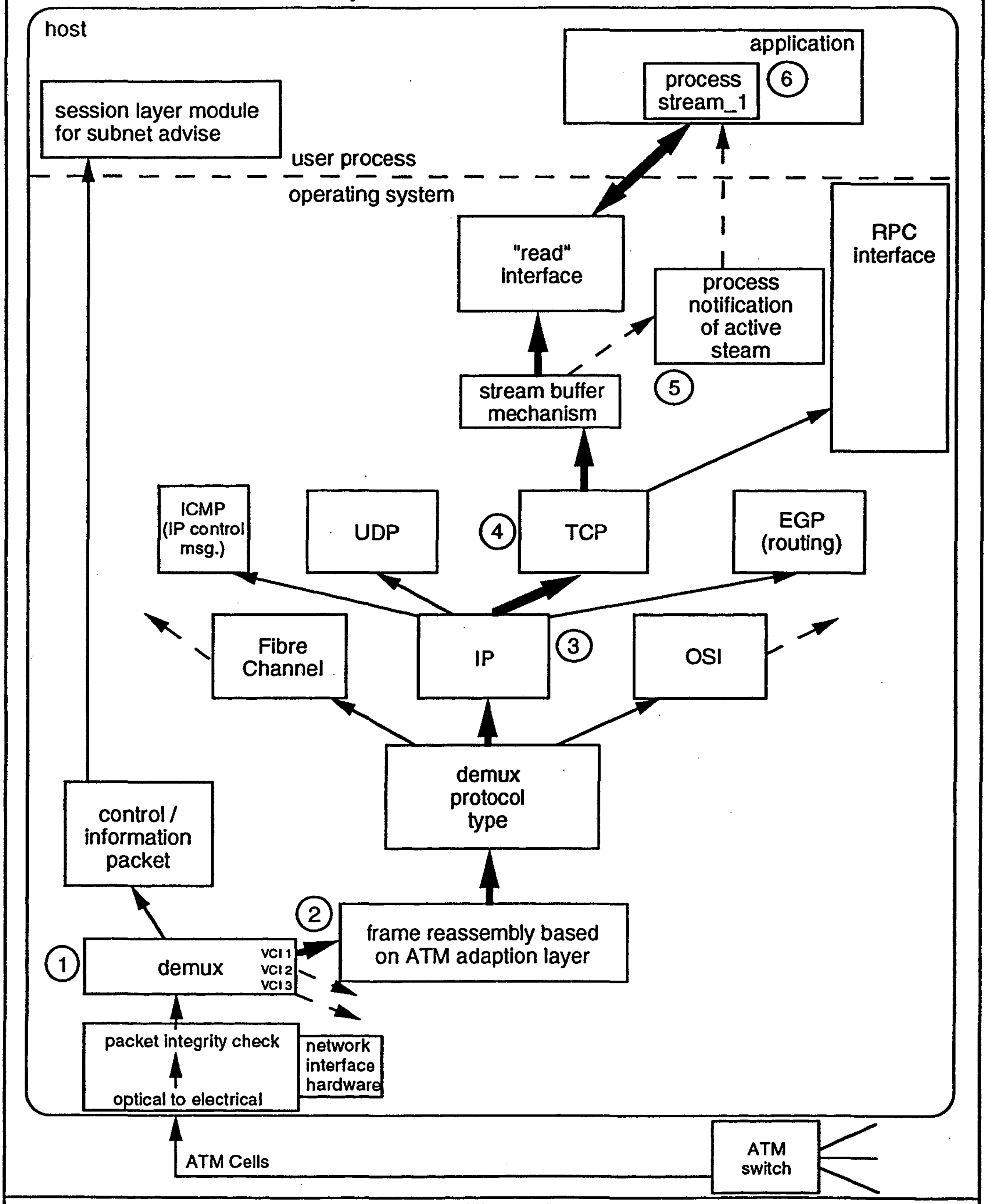


Figure 6

4.1. Distributed System Software Construction Paradigms and Interfaces

In constructing distributed systems much of the effort goes into the management and communications between the distributed software modules. PVM is a good example of a tool for assisting in this effort [Sunderam]. PVM is designed to facilitate the configuration of collections of computing elements into a parallel computing environment, but the step from that to handling a more diverse collection of configurable system elements is small.

Finally, once virtual systems are technically possible, in order for them to be useful in practice they must be easy for the information analyst to use. In the past several years "iconic" programming systems (e.g. AVS and Khoros) have demonstrated success in providing a high level programming "language" for non-programmers that can be used to specify data flows and interrelationships between functions graphically (in much the same way that the data flow diagramming CASE tools do).

4.1.1. The PVM System

PVM (Parallel Virtual Machine) is a publicly available software package that allows the utilization of a heterogeneous network of parallel and serial computers as a single computational resource. It provides facilities for initiation, communication, and synchronization of processes over a network of heterogeneous machines.

PVM may be implemented on a hardware base consisting of different machine architectures, including single CPU systems, vector machines, and multiprocessor machines. These computing elements may be interconnected by networks, and the hosts that are utilized may be varied dynamically. The PVM software essentially consists of a collection of parallel programming constructs to transfer data and synchronize the parallel operation of tasks. The support software interprets requests generated by the user-level constructs and carries out the necessary actions in a machine independent manner.

Application programs are composed of "components" that are subtasks at a moderately large level of granularity. During execution, multiple "instances" of each component may be initiated. These application programs view the PVM system as a general and flexible parallel computing resource that supports shared memory, message passing, and hybrid models of computation. Resources may be accessed at three different levels: the "transparent" mode in which component instances are automatically executed on the most appropriate hosts; the "architecture-dependent" mode in which the user may indicate specific architectures on which particular components are to execute; and the "low-level" mode in which a particular machine may be specified. Such layering permits flexibility while retaining the ability to exploit particular strengths of individual machines or architectures on the network. This provides a method for executing components on the host with an architecture (serial, parallel, vector, etc.) most suitable for a given algorithm.

HeNCE (Heterogeneous Network Computing Environment) [Beguelin] is an X-windows based software environment implemented on top of PVM, and is designed to assist in developing PVM applications. In HeNCE the program is specified as a graph, where the nodes in the graph represent procedures and the edges represent dependencies. These program graphs are entered using a graphical interface. The procedures represented by the nodes of the graph are written in C or Fortran, and in many cases these procedures can be taken from existing code. HeNCE provides facilities for editing and compiling these procedures on the various architectures of the user's defined virtual machine and then monitoring their operation. While PVM provides the low level tools for implementing parallel programs, HeNCE provides the programmer with a higher level abstraction for specifying parallelism.

4.1.2. AVS and Khoros

The Application Visualization System (AVS†) (a commercial product of AVS Inc., see [Upson]) and Khoros† (from the University of New Mexico) [Rasure] are examples of user interface paradigms designed to let non-programmers specify flexible data analysis and visualization "programs". These systems provide (1) a "visual programming" paradigm based on iconic ("picture block") construction of executable networks of predefined program "modules", (2) a software bus of sufficient sophistication to transport a wide range of scientific data between these modules, and (3) a general interaction mechanism that allows easy, interactive control over the operation of the data transformation and visualization aspects of these modules. Modules are combined into "networks" to perform a particular task. Networks of modules are easily modified via an X-windows interface. Data flow between modules is indicated by simply drawing a line between the module icons.

AVS contains a suite of modules for filtering, mapping, surface and volume rendering, animation, and data analysis. The AVS "flow executive" is the component that determines when to run various modules, provides both upstream and downstream data flow, and shared memory support. Khoros components include a visual programming language, code generators for extending the visual language and adding new application packages to the system, an interactive user interface editor, an interactive image display package, an extensive library of image processing, numerical analysis and signal processing routines, and 2D/3D plotting packages. Figure 7 shows a typical data analysis "program" put together with the AVS interface. Both AVS and Khoros contain the ability to distribute modules to various heterogeneous machines. Users can select the machine with the optimal architecture for each particular module.

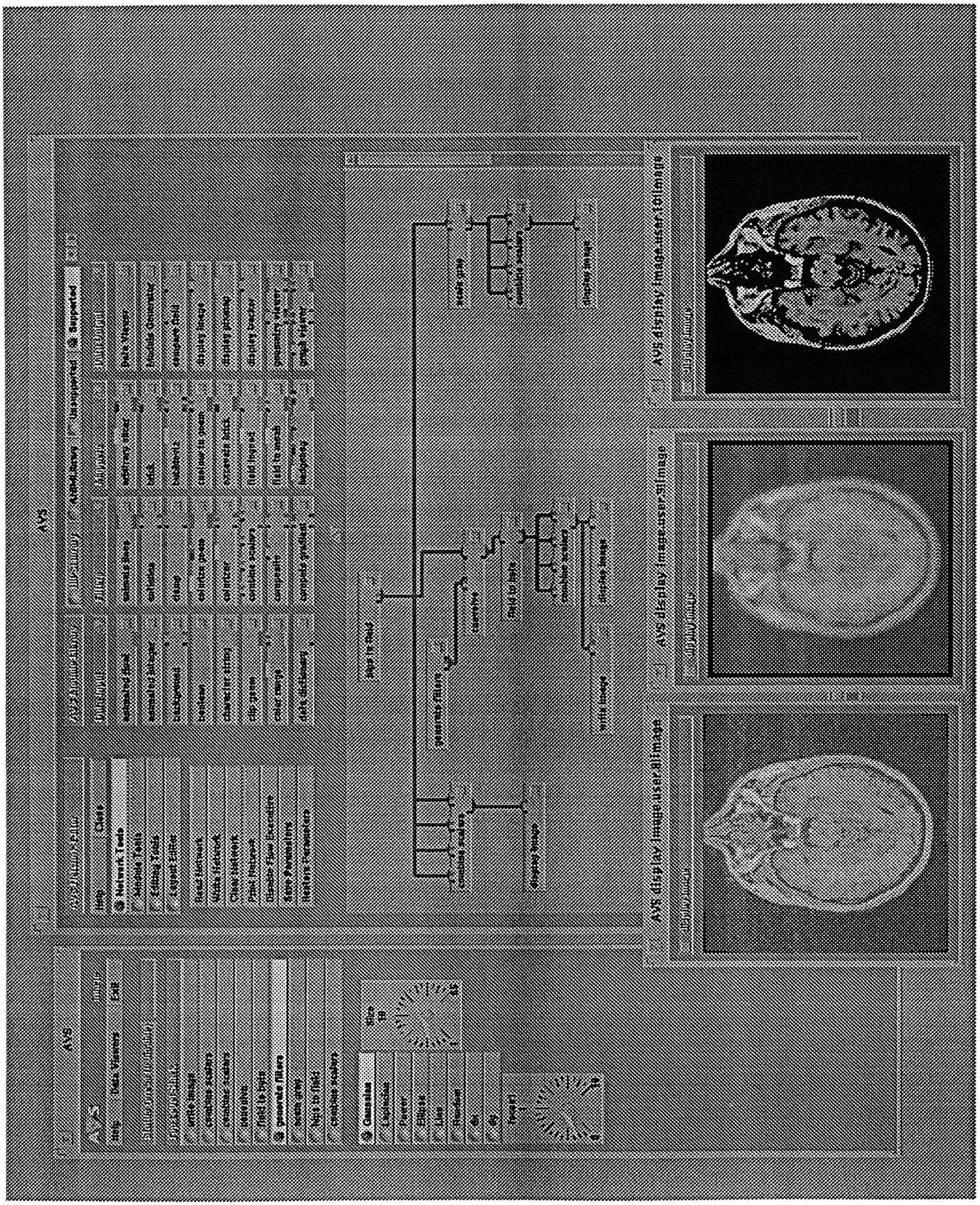

AVS
Help
Data Viewers
Exit
write image
combine scalars
combine scalars
convolve
field to byte
generate filters
scale grey
hips to field
combine scalars
Gaussian
Laplacian
Power
Ellipse
Line
Random
dx
dy
Size
10
1
65
focus1
1
0
10
AVS
Help
Close
Network Tools
Module Tools
Editing Tools
Layout Editor
Read Network
Write Network
Clear Network
Print Network
Disable Flow Executive
Save Parameters
Restore Parameters
Unsupported
Supported
animated float
animated integer
background
boolean
character string
clip geom
color range
data dictionary
animate lines
antialias
clamp
colorize geom
colorizer
combine scalars
composite
compute gradient
arbitrary slicer
brick
bubbleviz
contour to geom
excavate brick
field legend
field to mesh
hedgehog
Data Viewer
Module Generator
compare field
display image
display pixmap
display tracker
geometry viewer
graph viewer
hips to field
generate filters
combine scalars
convolve
field to byte
scale grey
combine scalars
display image
combine scalars
write image
display image
display image
AVS display image.user.8!image
display image
AVS display image.user.9!image
display image
AVS display image.user.10!image
display image

Figure 7

4.2. Network Based Computing System Elements

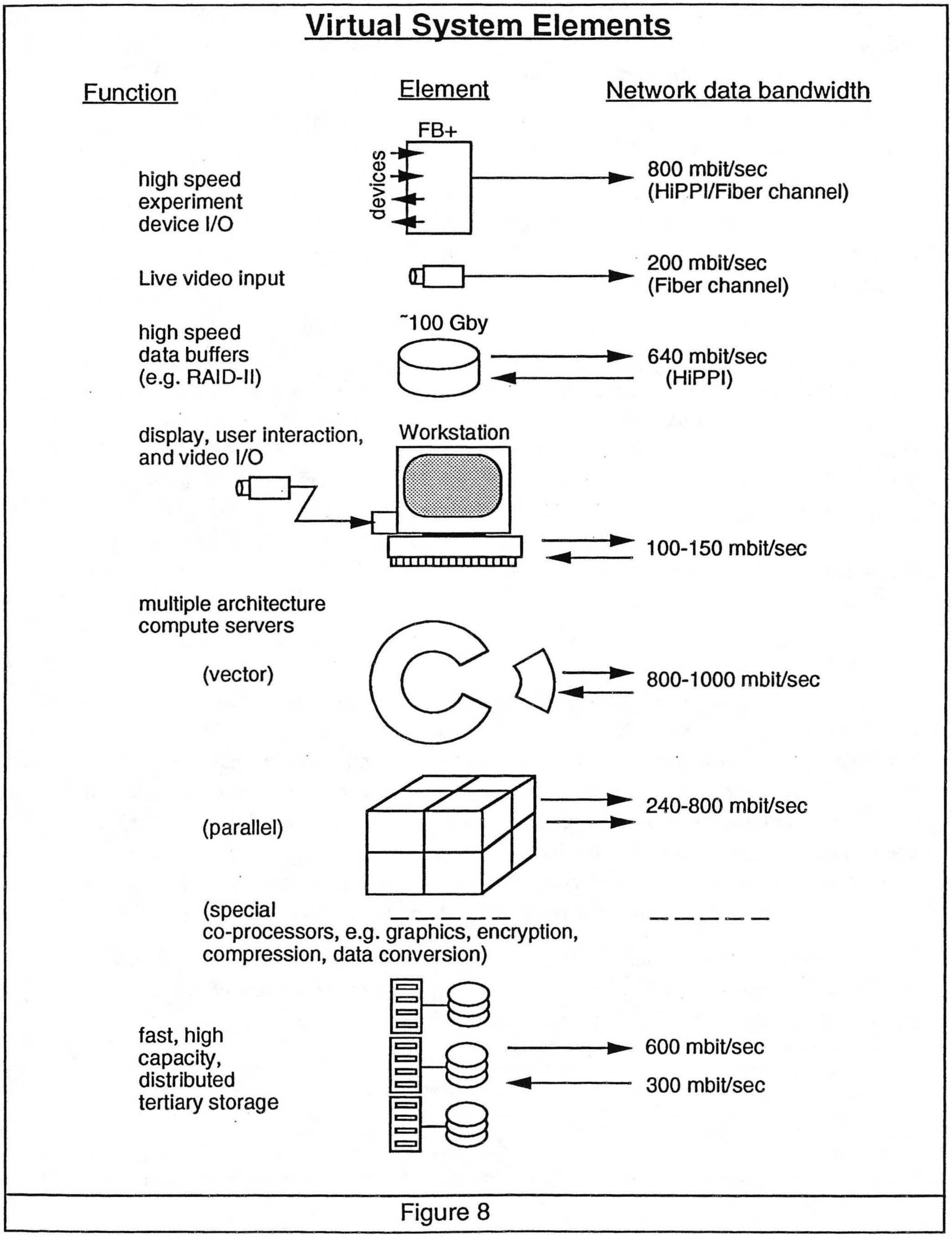
Given the mechanisms for interconnecting the computing and data handling elements that will provide us with our ''virtual computer system'' (or ''meta-computer'' as some people call it) we should consider what are the available elements. Figure 8 indicates the kind of elements that will be useful, and that we can expect to see in the next few years.

One of the important uses for configurable systems will be the remote siting of experiments. In order for this to be possible we will need to develop high speed instrumentation sub-systems that also have high speed network interfaces. This work is in progress in the form of prototype systems that handle certain aspects of physics particle-detector data analysis. Currently this work is focused on Fibre Channel[7] technology, but as ATM become better understood, it will undoubtedly also be used. In any event there is work in several of the gigabit testbeds to build adapters that will interface both Fibre Channel and HiPPI[8] to ATM.

Live video input devices (that is, network interfaces to analogue video sources) will also be important in the realm of experimental science. Video can provide for high speed image capture, for monitoring aspects of experimental apparatus that are difficult to monitor otherwise, and to provide backup for other monitoring systems. In the environment that we envision, video cameras will be scattered around the network as needed, and controlled remotely to enable video streams to the workstation.

When high speed (or ''real time'') data collection is necessary, it will almost always be necessary to buffer that data in order to decouple it from real time. This decoupling will be needed to allow time for processing and/or entry into a mass storage system. The likely candidate for this ''network buffer'' will be the new generation of RAID technology. RAID (Redundant Arrays of Inexpensive Disks) couples many disks together in a parallel fashion, providing for both high speed I/O and access through the mechanism of striping data across many disk units. RAID-II [Lee] is a network disk controller in which the ''file system'' information is maintained on an associated processor, but the data transfers take place directly between the disk array controller and a high speed network interface (currently HiPPI). Large RAID systems will be expensive, but being attached to a high speed network will permit sharing among multiple users and experiment data sources.

[7] Fibre Channel is a relatively recent effort that seeks to overcome some of the disadvantages of HiPPI. Fibre Channel is a serial line protocol (unlike HiPPI which has a 32 or 64 bit parallel physical implementation.) Like HiPPI, Fibre Channel is also a circuit switched technology, but both the switch and the host interface implementations are capable of switching fast enough to appear like a packet switch to the host. This makes it easier to do IP style, packet based networking using Fibre Channel, than it is with HiPPI.



Virtual System Elements

Function	Element	Network data bandwidth
high speed experiment device I/O	FB+ devices	800 mbit/sec (HiPPI/Fiber channel)
Live video input		200 mbit/sec (Fiber channel)
high speed data buffers (e.g. RAID-II)	˜100 Gby	640 mbit/sec (HiPPI)
display, user interaction, and video I/O	Workstation	100-150 mbit/sec
multiple architecture compute servers		
(vector)		800-1000 mbit/sec
(parallel)		240-800 mbit/sec
(special co-processors, e.g. graphics, encryption, compression, data conversion)	— — — — —	— — — — —
fast, high capacity, distributed tertiary storage		600 mbit/sec 300 mbit/sec

Figure 8

Workstations must also be connected into our virtual systems, typically as the support mechanism for the user interface. However workstations are also increasingly participating as computing elements of the virtual system through systems like PVM.

Supercomputer processing elements already participate in configurable systems, and an experiment in this area is described later in this paper.

Special purpose processors for graphics rendering, data conversion, data encryption, etc. will also likely participate in configurable systems. The ability to share these resources should also promote their development as a wider community of users will be able to access the hardware.

Finally, when all is said and done, data at various stages of collection and analysis must be archived. Widely connected high speed networks will again make possible the sharing of an expensive resource. However in the case of mass storage systems, networks will also enable a new generation of fully distributed storage systems (as envisioned in the IEEE Mass Storage System reference model, see [IEEE-MSS]). Further, these networks will permit the construction of higher speed mass storage systems by using the network to provide parallel access to multiple storage servers so that data can be striped across servers without a lot of special interconnect hardware. This approach is being explored in the MAGIC gigabit testbed. See [Catlett].

5. A Case Study

We now describe a test case application that is designed to exercise the various distributed computing mechanisms that will enable the assembly of virtual systems. The point of this test application is to illustrate the concepts, to help analyze the potential of currently available components that might be used in configurable systems, and to act as a tool for exploring the issues that inhibit the sort of routine, high speed interconnection that is needed to explore this environment. This application is designed to be useful in its own right, and demonstrates a fairly unique capability, but its primary purpose is as a tool to test the limits of configurable systems.

The prototype application is designed to achieve real-time distributed visualization of large data sets, and takes into account parts two and three of the imaging scenario described in Section 2 (connecting multiple remote computing elements and

[8] HiPPI (High Performance Parallel Interface) is a technology that is in relatively common use as a high speed local interconnect in the supercomputer environment. HiPPI "extenders" and Sonet interfaces are also being used to achieve very long distance (thousands of kilometers) point-to-point HiPPI links. Locally HiPPI is routed using circuit switches, and while the switches can switch circuits very quickly, the HiPPI host adaptors and software is generally very slow for connection set up and tear down. HiPPI is generally used to provide relatively long lived connections between supercomputers and their peripheral devices.

workstations), but not part one (on-line data sources), since the data set had already been generated and resides on storage local to one of the computing elements. The current T-3 NSF and DoE network environment allows sending images that have been rendered on a supercomputer to a local workstation at rates sufficient for visualization of complex 3D geometry (e.g., typically 10 320×320, eight-bit per pixel, uncompressed images per second). The full capability, which envisions real time transmission of data from imaging devices to a supercomputer, requires substantially higher bandwidth networks than available in T-3 based networks and is being investigated in several of the national gigabit testbeds.

The computational part of the application is partitioned into two pieces, one optimal for a massively parallel architecture, and one optimal for a vector processor. For this experiment the first part is run on a Thinking Machines† CM-2, and the second on a Cray†, Y-MP. The two compute elements are located at the Pittsburgh Supercomputer Center (PSC), and communicate with each other over a HiPPI, 800 Mbits/sec communications channel, while the remote workstations are connected to PSC via the usual variety of local, regional, and wide area networks (NSFNet and ESNet). (See Figure 9).

This application uses a virtual system configured from three elements, and demonstrates the use of wide area networks to enable access to high performance visualization by any remotely located scientist with a color X workstation. It also demonstrates the concept of using networks to assemble multiple computing elements whose different architectures optimize the solution of a specific problem.

5.1. The Application

The prototype application chosen to demonstrate the use of high speed networks with multiple supercomputers is the interactive display of MRI data of the human brain. "Interactive" here is taken to mean the ability to generate and display images at a rate of at least 3-5 frames per second as a result of changing the segmentation (region of interest) and at a rate of at least 10 frames/sec when changing the 3D viewing parameters for the resulting geometry (e.g. rotating). A typical MRI data set is 256 x 256 x 128 x 1 byte voxels, or 8.4 MByte of data.

The motivation for this particular use of the application is that a medical researcher would like to be able to interactively segment (isolate and identify) regions of interest, remove portions of the 3D data in order to "see" the resulting surface (i.e., dissection), view the results from various angles, and continuously rotate the results to generate 3D depth cues. The computational requirements necessary to accomplish this are still considerably beyond the ability of current scientific workstations, and in any event, one goal is to permit the end user to access the results from a low cost workstation. In previous experiments, we have found that the required performance is also

Case Study "Virtual System"

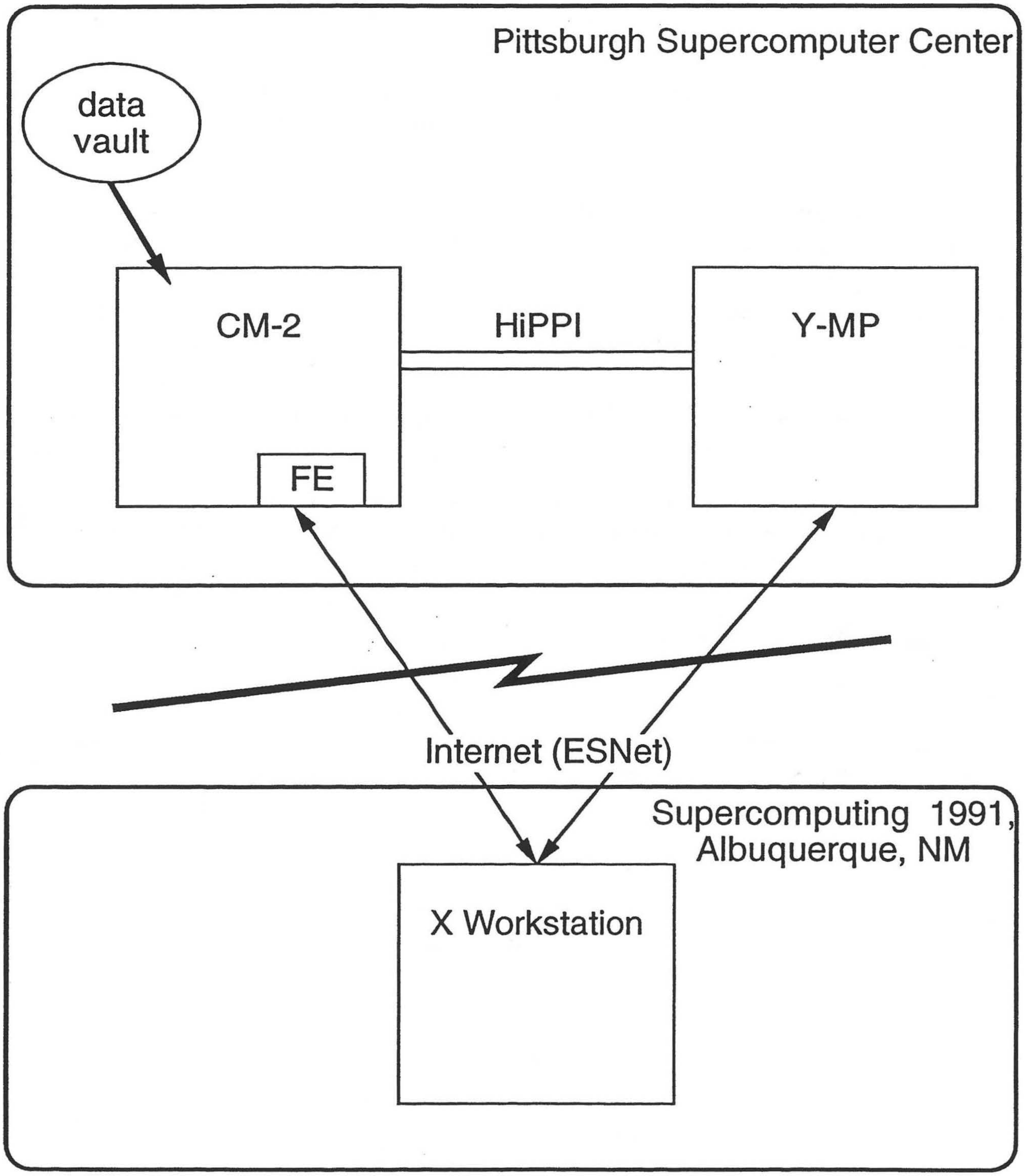

Pittsburgh Supercomputer Center
data vault
CM-2
FE
HiPPI
Y-MP
Internet (ESNet)
Supercomputing 1991, Albuquerque, NM
X Workstation

Figure 9

beyond the capability of a single Cray Y-MP processor. Our current goal is to show that interactive speeds can be achieved by distributing the computations across a collection of architecturally optimal supercomputers.

Figure 10 shows a screen dump of the operating application. The display is being generated on, and sent from the Cray, and the various control panels provide for interactive control of the servers on the CM-2 and the Cray.

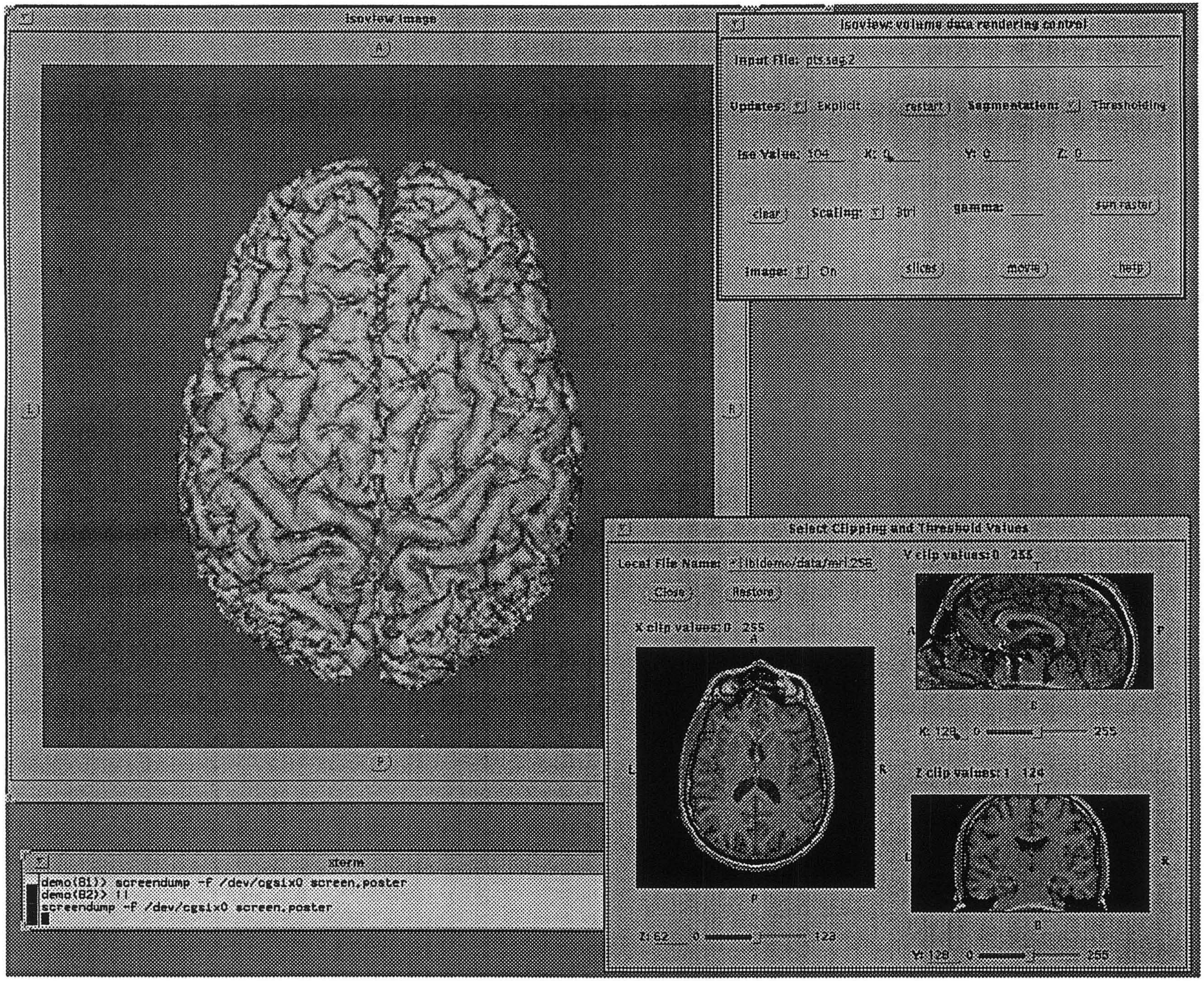
5.2. The Software Architecture of the Application

The prototype application is an example of the quantitative analysis and visualization of large 3D scalar fields. This process involves: (1) locating a region of interest (a contour surface and/or its enclosed volume); (2) representing that surface as geometry (a triangular tessellation, or 3D points on the surface), and; (3) the conversion of the geometry to a viewable image (viewing and rendering). (See Figure 2.)

A surface (rather than volume) rendering method is used for visualizing the region of interest. Surface rendering methods create a representation of an object surface using graphics primitives, such as polygons, which are then displayed through a traditional graphics pipeline. The idea is to "tile" the surface of the 3D object using thousands of small polygons or 3D point primitives. The first step in the creation of a geometric representation during surface rendering is to determine the location of the boundaries of the object of interest. This step is non-trivial due to the lack of distinct boundaries between various soft tissues represented in MRI data, and this makes it impossible to extract soft tissue features using simple thresholding. For example, the surface of the brain has an MRI value which is very similar to the skull and the membranes surrounding the brain. Our approach for this application was to have a human expert create a "mask" (outline) image of the surface of the brain. The mask image is a 3D binary data set, where a "1" indicates that a given location may be used in constructing the geometric representation of the brain. The process is semi-automated: the user inputs a small set of data points for each slice of the MRI data, and then a program running on a workstation performs local feature detection to generate an approximation to the desired mask. The user must then correct any problems in this mask by hand. Even with this partial automation the process is tedious, and more work is necessary to speed up mask generation.

Given the mask, simple thresholding is then used within the region of data defined by that mask. This ensures that no objects outside the surface defined by considering the masks of every slice (e.g. the surface of the brain) are ever part of the segmented object.

Marching cubes and *dividing cubes* [Cline] are two algorithms which are used to



Input File: pts.sag2

Updates: Explicit

Segmentation: Thresholding

Iso Value: 104 X: 0 Y: 0 Z: 0

Scaling: 3dl gamma:

Image: On

Local File Name: ~/libdemo/data/mri.256

X clip values: 0 255

Y clip values: 0 255

Z clip values: 1 124

Z: 62 0 123

X: 128 0 255

Y: 128 0 255

```
demo(B1)> screendump -f /dev/cgsix0 screen.poster
demo(B2)> !!
screendump -f /dev/cgsix0 screen.poster
```

Figure 10

create a 3D surface representation from a 3D volume dataset. In these algorithms, the volume of the voxel data is divided into "cubes", where each of the eight corners of a cube represents a voxel that is either inside or outside the surface representing the object to be rendered. If the voxel values of at least one, but not all, of the corners of a cube are greater than the surface threshold value, then that cube intersects the surface. In *marching cubes*, the surface intersecting the cube is represented by using up to four triangles. The algorithm computes the triangles for one cube, then "marches" to the next cube. The *dividing cubes* algorithm is similar to *marching cubes*, except that the output is 3D points with normals ("directed points") instead of triangles. *Dividing cubes* subdivides each surface voxel into small cubes that match the display pixel size, resampling the voxel to increase the apparent resolution. This approach eliminates the need for scan conversion of the triangles.

5.3. Method

The application is partitioned among three processes: a client and two servers (Figure 11). The client runs on a local workstation, and controls the operation of the three processes via an X-window interface. User input of parameters drives the computation performed on the servers. One server runs on the front end control system of the CM-2 (CM-FE). This server instructs the CM-2 to read the MRI data that has been placed on the Connection Machines' high speed file system (the Data Vault). The CM-2 performs the *dividing cubes* algorithm to generate a surface representation of the data. The resulting surface data is sent across a HiPPI channel to the second server, which runs on a Cray Y-MP. The Cray renders the 3D geometry into an image, which is sent across the network to the local workstation.

An analysis of the computing and communication involved is as follows. (The numeric annotation refers to Figure 12.) Greater detail on issues involved with each component of the application architecture is given in the following sections.

At client start-up time, the client establishes connections with the two servers, which are already running. After this initial connection, both servers then go into blocking reads until further data is received. Then,

(1) The user (U) specifies the file name of a data set that resides in the CM Data Vault. The CM-FE server causes the CM-2 to read in that file (1a) and performs preliminary computations (1b), returns data set information to the client, and then goes into a blocking read again.

(2) The user (U) inputs parameters that control surface generation (e.g., the isosurface value) and the view of the resulting surface (e.g., rotation about the x, y, and/or z axes). Once these are chosen, the user clicks a button to begin the process of computation. The client sends the viewing parameters to the

Case Study Application Architecture

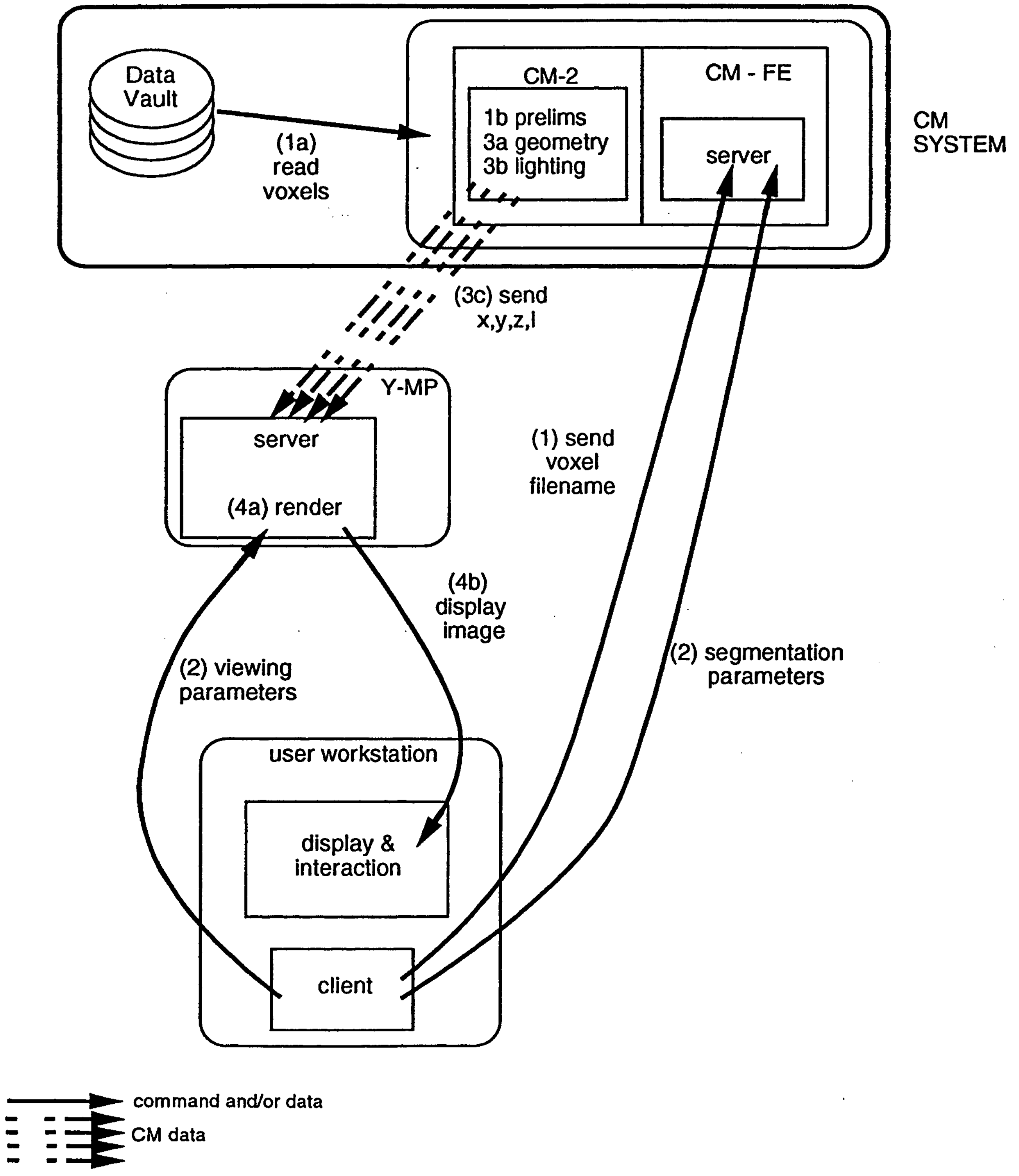


Figure 11

rendering server on the Cray. After receiving the parameters, the Cray goes into a blocking read on the HiPPI channel. The client also sends tessellation parameters to the server on the CM front end.

(3) On receipt of the tessellation parameters, the CM-2 generates the geometry (3a) (x, y, and z location of the surface) and then performs lighting calculations (3b), which generate an gray-level intensity for each point, and then sends the resulting four-byte quantity for every point over the HiPPI channel to the Cray (3c).

(4) Once the CM-2 has sent the data, then the Cray performs an orthographic projection (4a) (based on the current viewing parameters) and hidden surface removal (using a z buffer). The resulting image is then sent (4b) via TCP/IP to the client.

If the user had selected parameters to generate a single image, both servers have now gone back into blocking reads and are awaiting further parameter changes.

The user can also specify parameters that generate a sequence of images, e.g. gradually changing the viewpoint of the object. In this case, for the first image in the sequence steps 1 through 3 are followed as before. However, after sending the geometry and lighting data to the Cray (step 3c), the CM-2 server immediately starts calculations for the next image. Since the geometry is unchanged by resetting the viewpoint, all that the CM-2 need calculate are the new lighting values for the next image. At the same time, the Cray is rendering the current image and transmitting it to the workstation for display (see Figure 12). If the CM-2 finishes its computation before the Cray, it goes into a blocking write on the HiPPI channel. Thus in this mode computations for the CM-2 and Cray are parallel.

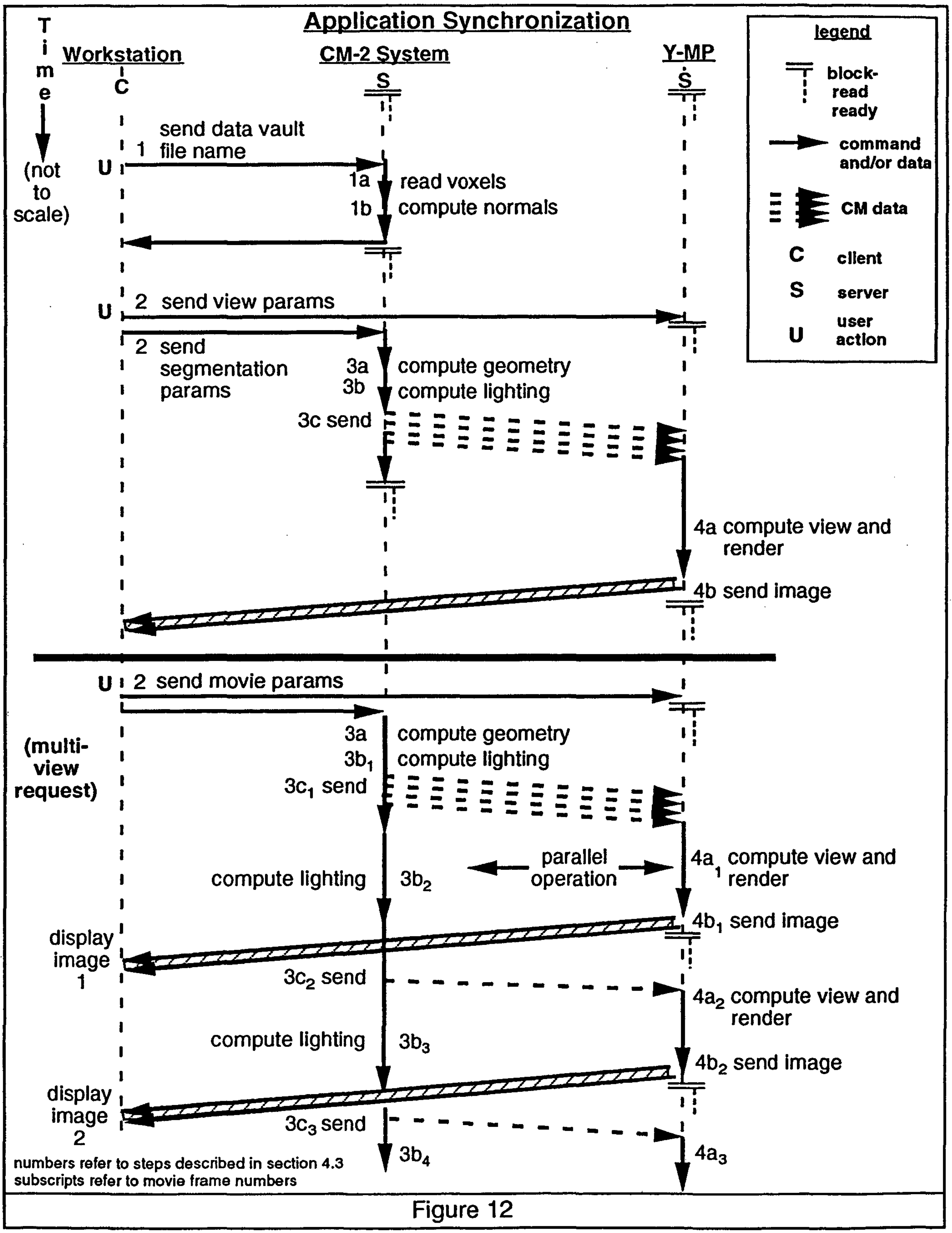
5.4. Issues in Mixed Computer Architecture Distributed Computing

In this section we analyze some of the issues that arise for the virtual system. The timings mentioned are for a data set of 8.4 million voxels.

5.4.1. Connection Machine Issues

Dividing cubes is a data-parallel algorithm, and hence maps very well to a large SIMD computer like the CM-2. Normal vectors must be found for selected voxels in order to do lighting calculations, and the calculations for finding the normal vector of every data point are performed in parallel immediately after the data is loaded into memory. This computation takes a few seconds, and only needs to be done once, since the CM-2 has sufficient memory to store all normals.

Determining which cubes intersect the surface can also be done in parallel, and takes about 0.11 seconds. The CM-2 processors can communicate with their



numbers refer to steps described in section 4.3
subscripts refer to movie frame numbers

Figure 12

neighboring processors very quickly, and this is the only type of communication needed to compute normals and surface locations. However the results are scattered across many processors, and must be gathered together before they can be sent to the Cray. This is done using a "send" operation, which takes about 0.7 seconds. After the results are gathered, the data is converted to Cray format and sent to the Cray server over the HiPPI channel.

5.4.2. HiPPI Channel Issues

Data is transmitted over the HiPPI channel utilizing a HiPPI specific protocol for high-speed parallel data transfer. The CM-2 is connected to the HiPPI channel via the CM I/O bus, a direct connection to the CM-2 supporting data rates of up to 240 Mbits/sec. Outbound data from the CM-2 is received from the CM I/O bus by the CM HiPPI station manager, a Sun-4† with special hardware for fast data movement, and is then placed on the HiPPI channel.

Because of the radical differences in machine architectures, data conversion between the CM-2 and the Y-MP is a non-trivial problem. On the Connection Machine, which uses many bit parallel processors, there is no straightforward relationship between the internal ordering of the data and the serial data ordering used on the Cray. The bits are ordered according to what Thinking Machines calls the "twiddle shuffle," a complicated geometrical reordering of data based on the underlying hypercube architecture of the CM-2. Our application must perform a conversion to serial data form for each block of data transferred. This conversion takes approximately 0.06 seconds for the 1.8 MByte of data typically transferred in one server operation. To simplify the use of the HiPPI channel, PSC has written a library of routines to hide both the networking and data conversion issues from the user. [Schneider]

5.4.3. Cray Issues

The main issues in using the Cray in an optimal manner are the amount of memory available to store the incoming data, and vectorizing as many of the calculations as possible. The Cray Y-MP at PSC has eight processors and 32 megawords of memory. This application uses only one of the eight processors, and a single user process only has access to four megawords of memory. This is not enough memory to store incoming x, y, z and normals vectors as 64-bit floating point numbers. Because of this, the lighting calculations are performed on the CM-2 instead of the Cray to reduce the amount of data to be transferred and stored. This approach eliminates the need to store the x, y, and z components of the normal for each point (each component would need at least 16 bits of resolution). Using the normals to perform the lighting calculation on the CM-2, all that need be sent is a resulting 8-bit intensity value, a 6-fold reduction in data volume. (Transferring an 8-bit value also reduces the amount of data conversion necessary.)

Using an 8-bit data type brings up the problem that that the Cray is a word-oriented rather than a byte-oriented architecture, and the use of character data types in a loop inhibits vectorization. To circumvent this problem, the data is read off the HiPPI channel in integer format and the data for two points are placed in one Cray word. Floating point conversion using bit shifting operations to "unpack" the stored data vectorize readily, and take up a negligible portion of the total calculation time. Calculations are performed on groups of points that have been converted into floating point format. This conversion must take place for every frame, since as noted above, there is insufficient memory to store all points in floating point format.

Since the lighting has been performed on the Connection Machine, the portion of the rendering that the Cray must perform is projection and hidden-surface removal. The projection calculation vectorizes trivially. Vectorizing the hidden-surface removal is more difficult, and we have implemented two hidden-surface removal methods as a result. One, using a z-buffer approach [Foley], is very general. Any view of the data can be taken, and the data can arrive from the Connection Machine in any order. However, to achieve this generality, it is almost impossible to safely vectorize the code implementing this method.

We have also implemented a much faster but less general method for viewing the data, using the painter's algorithm [Foley]. Rotation may only be around one axis, and the data must be pre-sorted on the Connection Machine in order for this method to be accurate. This pre-sort takes several seconds, but only needs to be done once for a particular surface. Using this technique, rendering is more than ten times faster than when using the z-buffer. For a typical data set of several hundred thousand points, the maximum viewing rate using the z-buffer is around three frames per second, but using the painter's algorithm we can render over 30 frames per second. (This speed is not achieved at the workstation because of network bandwidth restrictions between the distributed components in this application).

5.4.4. Local Workstation Issues

One of the main reasons for performing the visualization on supercomputers instead of a local workstation is to demonstrate that scientists can use a configurable environment to assemble available computing elements in order to do sophisticated visualization of large data sets, and then display the results on an inexpensive color workstation.

For this experiment to be successful it is necessary to send raster images between the rendering server (the Cray) and the image display device (the workstation) fast enough to achieve motion/rotation visual queuing (5-10 frames/second, minimum). This experiment is designed to test the new 45 Mbits/sec wide area networks and 100 Mbits/sec (FDDI) LAN's and workstation interfaces.

Another workstation issue arises from the display of images using the X protocol when those images are coming in at interactive rates. This demonstration uses an X-Window server, together with with Sun's XView† toolkit (to handle the user interface) and a proprietary direct graphics access library (Sun Microsystems' XGL) in order to display the images at a fast enough rate. (For displaying images the overhead of the standard X protocol causes a noticeable degradation in image display speed.) This image display issue was made even more critical because typically images need to be zoomed by a factor of two or three for display. (In our implementation of *dividing cubes*, we do not perform voxel subdivision. Thus, image resolution is chosen so as to match voxel resolution. This typically results in images that are on the order of only 300×300 pixels.) The approach of using direct access to the workstation frame buffer through a low level interface that coordinates with the window system to set up the screen display area, but does not use the window system image display mechanism, is fairly common, and most workstation vendors provide similar mechanisms.

5.4.5. Network Protocol Issues

One of the issues encountered early on was the problem of using TCP in a high speed, wide area network environment. The problem was that as network speeds increase, the throughput became limited by the speed-of-light propagation time between the communicating computers coupled with peculiarities in the TCP implementation. The standard version of TCP/IP can send at most 64KB of data per round-trip-time (twice the propagation time) and, in practice, the sustained throughput was at most half this theoretical maximum because of the packet acknowledgement scheme.

Working closely with Cray Research and Sun Microsystems, Van Jacobson has implemented the TCP/IP extensions described in RFC1072 [Jacobson88] and RFC1185 [Jacobson90]. These extensions remove the 64KB per round-trip-time limit and allow TCP/IP to run at the full speed of the underlying network, independent of the end-to-end propagation time. The issues and techniques for this situation are described in [Jacobson88a]. These modifications are essential to achieving fast image display on the workstation.

5.5. Analysis

The analysis of the operation of this distributed system is summarized in Figure 13 and Table 1. Typically, 480,000 3D points are generated for the particular data set used. This process takes approximately 0.8 seconds on the CM-2 (time for surface generation plus time to gather results). From there, it takes 0.06 seconds for the data conversion, 0.1 seconds for the HiPPI transfer, and 0.3 seconds for the Cray to render the image (using the z-buffer approach). Over a 45 Mbits/sec network, it takes about 0.1 seconds to transfer the resulting 320×320×1Byte (100 KByte) image to the local

Analysis of the Application Time Budget

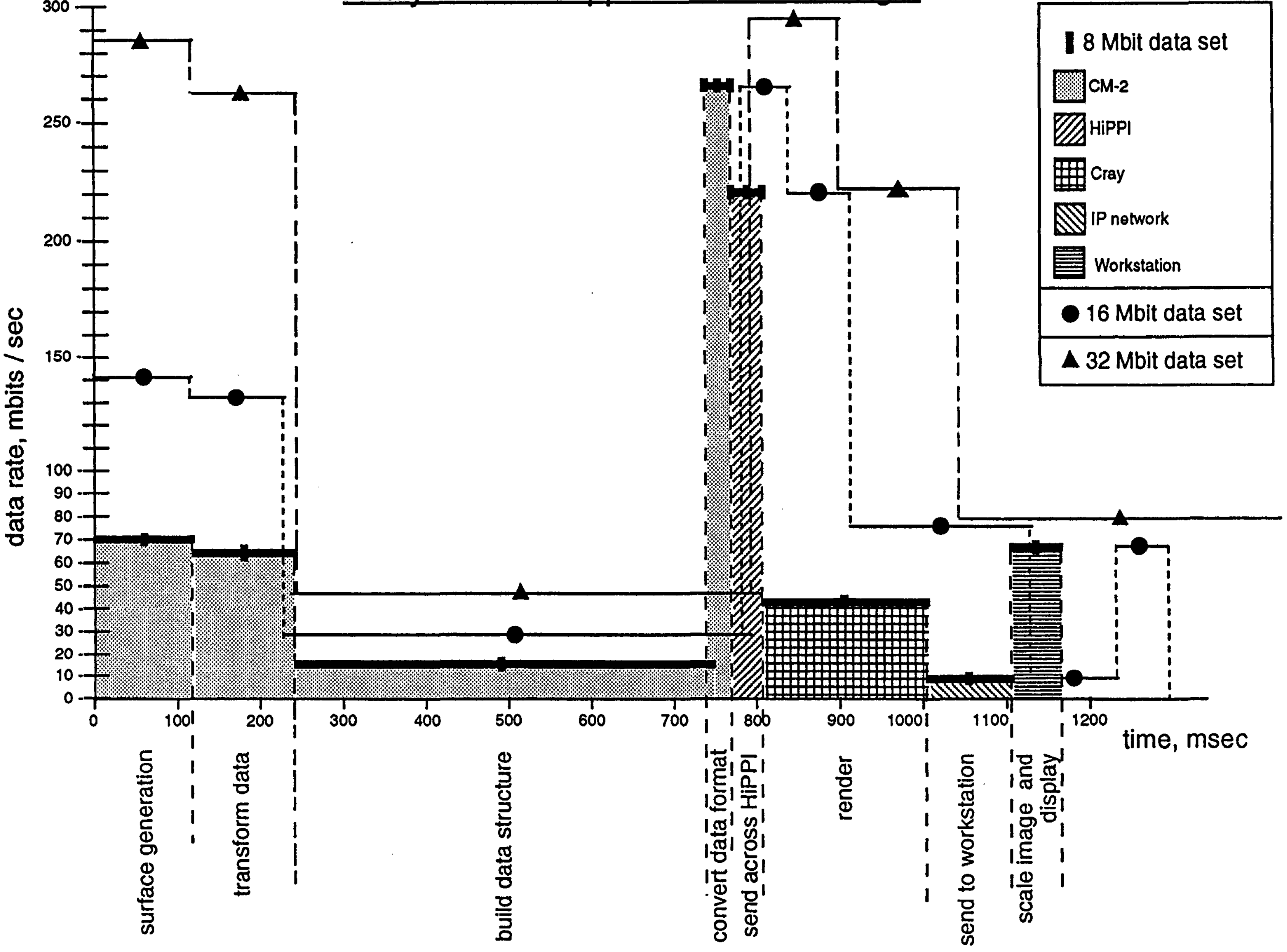


Figure 13

TIME BUDGET: CM-2/HiPPI/Y-MP/Inet/Display

256 x 256 x 128 MRI data (8×10^6 voxels)	CM-2		2 sequencers, 16k processors 500 MBy memory
Calculate normals	1.95		initialization
"Dividing Cubes"	0.11	rotation only mode	
gather 0.5×10^6 , 3D points	0.7		gather is a function of the number of processors that have data selected
lighting	0.01	0.01	
convert to serial data	0.06	0.02	
send 2×10^6 Bytes over the HiPPI channel	0.1	0.05	in rotation only mode send 0.5 Mbytes
CM total	0.98	0.08	
0.5×10^6 point + Intens.	Y-MP		1 processor, 4MW (32 MBy)
unpack & convert to floating point	0.01	0.01	
viewing & projection	0.35	0.3	z buffer algorithm
hidden surface removal		(0.03)	painter's algorithm
Cray total	0.36	0.3	
send 320 x 320 image (100 KBytes) over T3 link	0.1	0.1	45 mbit w/Jacobson TCP
	Workstation		
write image to display window	0.03	0.03	X + Sun's XGL
Grand Total	1.47	0.52	
movie mode: CM, Cray, & net I/O occurs in parallel		0.3 (0.1)	z buffer algorithm painter's algorithm

TABLE 1

workstation. Therefore the total time is around 1.3 seconds. Several tests were run varying the size of the input data set. Figure 13 shows the results for the cases of 8, 16, and 32 Mbit data sets, in terms of the ''bandwidth'' of each major operation. As is to be expected, the time for the data parallel operations on the CM-2 were independent of the amount of data being processed (within the memory size limit), and the vector operations on the Cray increased in throughput and required somewhat more time (though not nearly linear increase, since the longer vectors took better advantage of the processor). The case of the 64 Mbit data set continued the trend, but was not included in Figure 13 in order to make the graph easier to interpret.

In the general case the current performance does not yet meet our goal of five frames per second, but it is fast enough to be useful. Rotation of the geometric model representing the surface of the brain is faster because the geometry does not need to be recomputed. Rotation also requires only new lighting intensity values, which are only one-fourth the amount of data of the general case. In this case the CM-2 uses 0.01 seconds, data conversion takes 0.02 seconds, the HiPPI transfer takes 0.05 seconds, and the same amount of time as above for the Cray rendering and the image transfer. The total time in this case should be around 0.5 seconds. However, the application can be run in ''movie'' mode, where several images are generated from one set of parameters. In this case the CM-2 and Y-MP are working simultaneously, and can rotate images at a rate of about three frames per second.

The speed of rotation can be further increased, at the cost of generality, by using the painter's algorithm for hidden-surface removal as described in section 5.4.3. Rendering in this case takes 0.03 seconds. Thus in movie mode, network bandwidth becomes the limiting factor, and the maximum speed of rotation is ten frames per second.

One issue that has not yet been addressed is whether the use of supercomputer resources in this example is an efficient one. For example, would it be better to avoid the complexity and expense of using both the Connection Machine and the Cray, and perform all calculations on only one of these machines. In both of these alternative cases we would have had to use a larger portion of that resource than was utilized in our application to be able to achieve the same performance. In our implementation we use one Cray Y-MP processor that has essentially been dedicated to our use (our process is given the highest priority). This is necessary because, depending on the load of the Cray, it may take four seconds for a typical process to get swapped in, significantly reducing the speed of this application. Alternatively, we would have had to use more than one non-dedicated processor to get the same performance, but again, running the application interactively would be difficult, and there would be additional overhead. It can of course be argued that the typical user is not going to get highest priority on a process, but it is more of an administrative issue as to whether one dedicated processor, or more than one undedicated processors are more expensive for one user.

Alternatively, we could have performed all of the calculations on the Connection Machine. In our approach, we used two out of the four sequencers available. We could probably achieve the same or better performance using all four sequencers and performing all of the operations on the CM, but again, it is not feasible for a single user to use an entire supercomputer at a busy supercomputer center.

Another issue to be addressed in heterogeneous supercomputing is the amount of programming skill necessary to take full advantage of it. The current application made use of IPC socket-level programming to handle the communication between the components, but the need for low-level handling of communications should be eliminated as systems such as PVM become available.

The difficulty in writing parallel code is also an issue. Portions of the code that would be roughly equivalent in speed on both the CM and the Cray, but which are difficult to code using the SIMD model, were implemented on the Cray. Thus it was actually easier to program the application using both vector and massively parallel supercomputers.

6. Conclusions

We have presented a collection of technologies that, taken together, will provide the possibility for: (1) partitioning problems between heterogeneous supercomputers; (2) doing remote siting of data intensive scientific experiments; (3) providing access to capabilities that could previously only be obtained at a small number of sites due to the size, cost, or experimental nature of the implementation, and; (4) providing new capability enabled by the nature of the networks themselves.

The example application demonstrates that wide-area networks are no longer the bottleneck to the type of distributed imaging applications envisioned above. Of the many scientific imaging scenarios that require the described architecture, medical research can provide a clear and immediate focus to drive and justify the computing development, and demonstrate the utility and importance of an integrated, distributed, high speed computing environment. High speed network testbeds and associated prototype applications are the first step in the process of enabling this technology.

Widely deployed gigabit-per-second wide-area networks and the associated interconnecting hardware and software promise to alter the way that many large scale problems are approached. These networks will allow the creation of ''network'' or ''virtual'' supercomputers- computing systems comprised of geographically distributed components communicating with each other at high speeds, and configured on demand into virtual systems that exist only as long as necessary to solve a particular problem, or until a better combination of elements to solve the problem becomes apparent, at which point the virtual system is reconfigured.

7. Appendix A: Internet Routing

When the sending and receiving hosts are not on the same subnet (LAN), then the key to the functioning of the Internet is routing. (See Figure A1 for a typical Internet situation, and see [Narten] for a very readable account of routing and routing issues.) In figuring out how to get data from one system to another, the problem can be divided into two cases. The first is where two systems are connected to the same "link-level" network (i.e. a local area network, or "subnet" in ISO terminology). When packets are moved across a physical medium from one system to another there are always at least two levels of addressing involved: the globally known Internet address of the destination hosts, and its physical, or interface address. When packets are sent, they go from one host interface to another, and at the lowest level the sender must know the hardware address of the receiver's interface in order to do the sending. In this case the routing is typically accomplished by a table lookup that just returns the physical address corresponding to the Internet (IP) address of the destination. In this case the problem is to find the interface address of the target host. On most LANs the Internet mechanism for this is a special protocol called the address resolution protocol (ARP). ARP is an example of the resource discovery problem. When host A wishes to talk to host B (which is on the same LAN), A discovers B's physical address by broadcasting a link-layer packet that contains the IP address of the intended receiver and that of the sender. Since the packet is broadcast, all systems attached to the LAN receive the packet. Every system examines the packet, and upon finding that it is an ARP request, looks at the IP address of the intended receiver. If the current system is not the intended receiver, the packet is discarded, and no further action is taken. On the one system that is the intended receiver, the packet is updated to contain the physical address of the receiver, and sent back to the sender (the sender's physical address is also contained in the packet.) From this the sender makes an entry in its ARP table that maps the IP address of the target host to its physical address, and data communication at the IP subnetwork level can commence.

In the second case, the target system (say host F) is not on the local network. In this case the sender has to have a "route", or proxy destination, to send the packet to. In the simplest case, there will be a single "default" route for all hosts not on the local subnet. This route is really the address of a system on the local network that all non-local IP packets can be sent to, and this "router" will worry about how to get them to their destination. The router, of course, will typically have interfaces on several networks. If the target host is on one of the networks attached to the router (as might be the case in going from one campus department to another) then the process described above repeats itself, with the router acting as the proxy for the original system. Once the router knows the physical address of the target host, it sends the packets

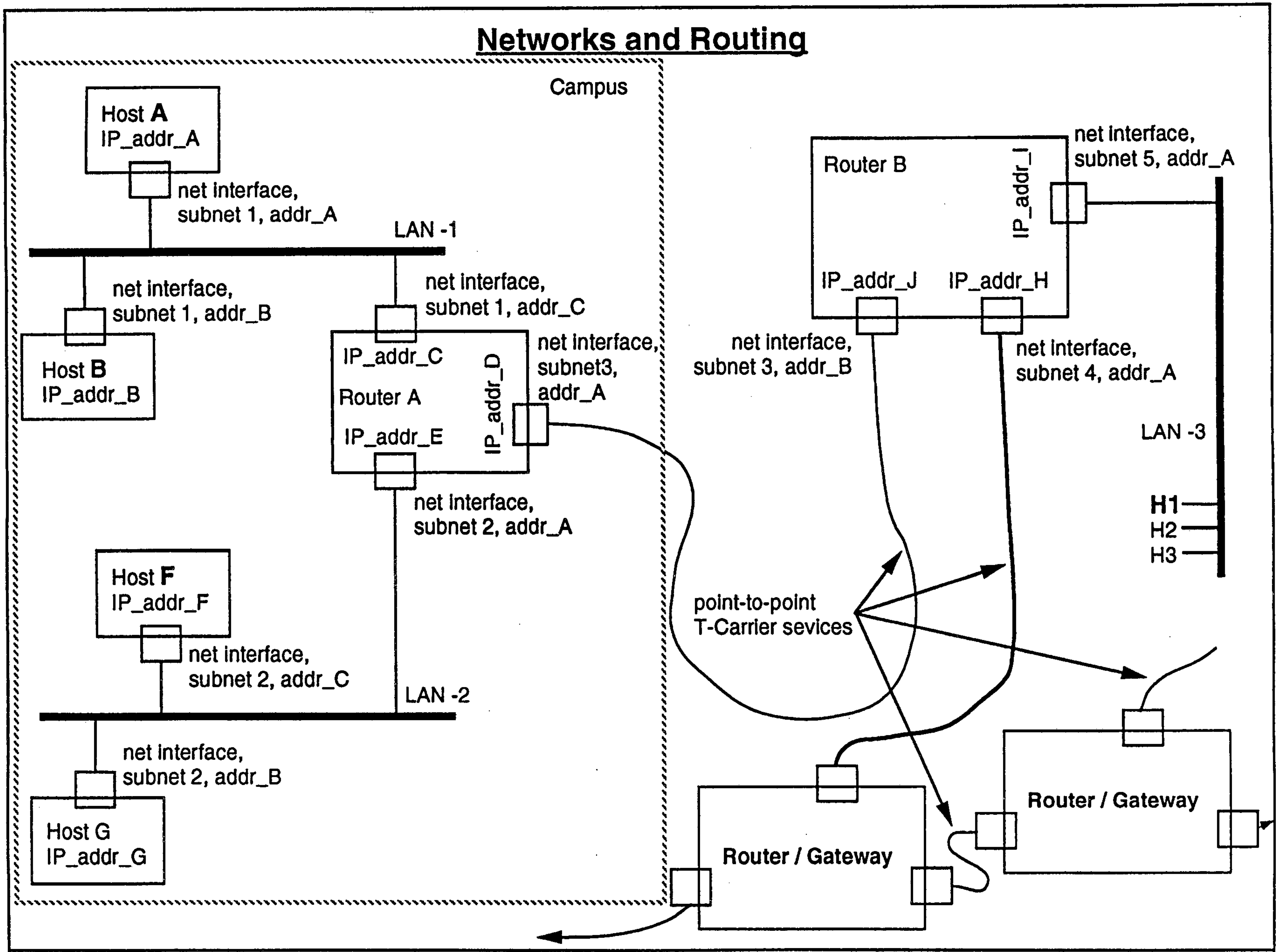

Networks and Routing
Campus
Host A
IP_addr_A
net interface,
subnet 1, addr_A
LAN -1
net interface,
subnet 1, addr_B
Host B
IP_addr_B
net interface,
subnet 1, addr_C
IP_addr_C
Router A
IP_addr_D
IP_addr_E
net interface,
subnet3,
addr_A
net interface,
subnet 2, addr_A
Host F
IP_addr_F
net interface,
subnet 2, addr_C
LAN -2
net interface,
subnet 2, addr_B
Host G
IP_addr_G
Router B
IP_addr_I
net interface,
subnet 5, addr_A
IP_addr_J
IP_addr_H
net interface,
subnet 3, addr_B
net interface,
subnet 4, addr_A
LAN -3
H1
H2
H3
point-to-point
T-Carrier sevices
Router / Gateway
Router / Gateway

Figure A1

to that system. For all future packets, the router just picks up the packets incoming from A from the router's network interface on A's subnet (LAN-1), and sends them out through the interface of the subnet (LAN-2) that B is connected to, and hence to F. Now assume that the target host (say host H1 on LAN-3) is not on any of the subnets directly connected to the router. In this case the router has to forward the packet to another router (presumably one "closer" to the destination), the second router forwards to another router even closer to the destination, and so forth until the packet arrives at a router that does have an interface on the subnet of the target host. The trick is how do the intermediate routers know what to do with the packets that cannot be delivered on one of their subnets. In other words, how does one router find the next router that is closer to the destination? The answer is that there has to be cooperation among the routers to determine how to move a packet through the Internet so that it eventually reaches its destination. Furthermore, the routes (the tables in each intermediate system that say how to get a packet closer to its destination) should (ideally) be determined so that the time to go from A to H1 is minimized. Routes should also be dynamic so that conditions in the Internet such as failed or congested paths are avoided. This is all further complicated by the circumstance of administrative policy. For example, certain paths may only be available to transport packets that originate or are destined for subnets of certain organizations (because the governing bodies of those organizations own the links). The problem of discovering the optimal routes through a large and non-isotropic network has been studied for a long time, and will be for a long time to come. There are routing protocols that provide mechanisms to go out and explore the network to determine the state of connectivity and traffic distribution, and then put together routing configurations and send them around the network for the routers to use. It is worth noting that one of the details that has been ignored in this brief discussion is that the LANs attached to a router are frequently not the same technology (e.g. FDDI and Ethernet), and may not carry the same amount of data in their (link level) packets. This can cause the original IP packets to be fragmented several times in the course of their transit through the Internet.

In slight variance to what was said above, routing protocols actually occur at two levels. The level described above (intra-domain routing) operates within a collection of routers that are within a single "administrative domain" (AD) (like a campus, or a regional network). A different set of protocols operates between ADs.

Administrative Domains (ADs) are the logical entities responsible for organizing routing for a collection of networks. Usually, an AD is organized around some other unit of organization, such as a university campus or an agency network such as DOE's ESNet. Within the Internet today there are at least 1000 ADs. The AD abstraction simplifies the problem of routing in the Internet to the problem of routing within each AD (intra-domain routing), and then routing between ADs (inter-domain routing). Within an AD, an organization runs at least one (often many) Interior Gateway routing protocols (IGP) to determine optimal paths through the network. Examples of IGPs are the Gateway-Gateway protocol (GGP), Routing Information Protocol (RIP), and HELLO, just to mention some of the names. IGPs allow routers inside an AD to calculate the routing tables for the AD's hosts and routers. ADs use at least one (often more) Inter Domain Routing Protocol (IDRP) to communicate between the ADs. The Exterior Gateway Protocol (EGP) is an example of an IDRP. See [Comer] and [Narten] for more information on this topic.

8. Appendix B: The OSI Model of Networking

The OSI model of networking is useful both for the standard terminology that it makes available, and because, at the lower layers, it provides a point of commonality with the telecommunications industry. The model is based on functional layering, and the notion (like the Internet model) that there are points in the hierarchy where elements of the information are the same on both sides of a connection. (This is essentially the same as saying that on every participating host there are certain well defined interfaces in the IPC mechanism, independent of any part of the underlying implementation, where everything is constant and well understood. It is, however, an interesting feature of internetworking that there may be many transformations as data works its way from one system to another, and that at intermediate points (typically in gateways) this situation may not hold.)

This section introduces terminology that is used by the telecommunications industry, and as the computing and telecommunications industries are drawn together (e.g. in the gigabit testbeds), is increasingly showing up in the world of distributed computing applications. The OSI model consists of seven layers (and is sometimes jokingly referred to as the seven layer cake).

Application (Level 7):

This layer can be user level software, but it is usually considered to represent the high level services that a user application is built on. Examples include data management and access mechanisms, directory (data lookup or dictionary) services, the authentication aspects of security, resource management and recovery, program instantiation (command interpreters), etc. Each of these will have a standard application programming interface (API).

The next two layers provide a core set of services for end-to-end data exchange. These services allow machine independent data exchange between heterogeneous systems. They are sometimes referred to as syntax independent data exchange services.

Presentation (Level 6):

This layer permits a common understanding of data representation formats, and structured data exchange. An integer is an integer and a floating point number is a floating point number at this interface, regardless of how differently the host systems at each end of a conversion might choose for the binary representations. The ISO, ASN.1 (Abstract Syntax Notation 1), and Sun's eXternal Data Representation (XDR) are both examples of facilities providing these services. These services not only have to do data conversion, but provide for serializing data structures for transmission across a network and reconstruction at the receiver. It is not uncommon for these operations to be by far the most expensive parts of IPC. The encryption aspects of security are also handled at this layer.

Session (Level 5):

This layer handles the coordination of dialogue between communicating applications. This can include things like full duplex messaging protocols and remote procedure calls. This layer also performs some connection management, and in the future might include the interfaces for communicating with network management facilities of the type envisioned in ATM networks for requesting quality of service, informing the network of upcoming usage, etc.

The boundary between layers four and five (sometimes called the transport service boundary) is the point at which network independent data exchange occurs. This is typically the lowest level in the architecture that user level processes (as opposed to operating system functions) have access to networking.

Transport (Level 4):

This layer does end-to-end message transfer, error control, fragmentation, and flow control. At this layer we find things like IBM's SNA (Systems Network Architecture), the OSI TPc (Transport Protocol classes), the Internet TCP (transmission control protocol), and the IBM PC's NetBIOS.

Networking (Level 3):

Network routing and addressing occur here. For example IP routes messages across multiple subnets in connectionless fashion. It uses the services of each subnet that it routes through to actually move the data, which may entail further fragmenting the data in order to meet the constraints of the subnet. Subnet routing in the form of protocols like ARP and ICMP also show up in this layer. The ISO, X.25 connection oriented protocol encompasses everything from this layer down.

Data Link (Level 2):

The link layer provides subnet access services to the network layer. The IEEE 802.x type LANs are examples of link level services. 802.2 (logical link control - LLC) provides common link layer control functions to the media access (MAC) layer. The LLC functions provide for multiple concurrent logical links via connectionless, connection oriented, or acknowledged connectionless transport. TCP/IP does not use 802.2 because, in the view of the Internet model, these services are redundant with functions provided higher in the protocol stack. The lower part of level two provides the physical link access. The functions are frame delimiting and error checking. Examples of MAC protocols are 802.3 (Ethernet), 802.4 (token bus), 802.5 (token ring), X3T9.5 (FDDI), and recently 802.6 (SMDS - a metropolitan area network).

Physical (Level 1): This layer defines the physical and mechanical connections to the signaling media.

There are a large number of terms associated with the OSI model of networking. It is useful to introduce a few of these:

Peer entity is unit that sends information to and from the same places in the architectural hierarchy.

Service Access Point is the place or means by which an entity accesses the services of a layer. Common examples are PSAPs (access to the presentation layer, TSAPs (access to the transport layer), NSAPs (the network layer) and LSAPs (the link layer).

Logical Link is a connection between a pair of network layer peer entities. LLs usually link peer entities in hosts on the same subnet, but can extend across link level bridges. LLs are the lower level equivalent of the upper level end-to-end connections.

Having covered the ISO model very briefly, the interested reader is referred to [Cysper] for more information about this, and the relationship between OSI, SNA, and TCP/IP networking.

9. Appendix C: Common-Carrier Services

The signal carrying facilities in the Internet have traditionally been the only part of computer networking provided by the common carriers. While this continues to be true, increasingly the common carrier services are moving up the OSI architectural hierarchy as deregulation allows the offering of more value added services. In particular, very high speed networks will almost certainly be provided as broadband Integrated Services Data Network (B-ISDN) services (mostly through the ATM packet network mechanism described below). This is at least in part due to the increased cooperation between the computing and telecommunications industry, and in part because the capability is so expensive that a cooperative effort is required. We now briefly describe the common carrier services.

T-Carrier Systems (OSI Level 1)

The original common carrier interoffice digital transmission systems were called T-Carrier Systems, some of which were eventually offered to end-users on a tariffed basis. The most common of these offerings are best known by their T-Carrier designations of T1 (~1.5 Mbits/sec) and T3 (~45 Mbits/sec). They are also commonly called by their digital signaling number representing their respective bit rate, i.e., DS1 and DS3. Note that DS0 refers to 56 Kbits/sec service as an incremental sub-unit of DS1 service.

These circuit services provide a synchronous transmission service between two points for almost any distance, as long as there are T-Carrier services available between the points. Most of the connections between routers at geographically distinct sites in the Internet are provided by T-Carrier services. In the past several years there has been a marked increase in the availability of, and a marked decrease in the costs for T1 circuits nationwide. This has helped to fuel the rapid expansion of the Internet national backbone networks as well as the regional/mid-level networks. Due to this, almost all communication equipment vendors now support T1 interfaces. Many local network environments are now routinely expanded through common carriers with T1 circuits using commercially available termination equipment. This class of equipment and circuits, considered quite exotic just two to three years ago, is now in routine use and is no longer considered operationally complex.

T3 circuits are now beginning to become available on a more widespread basis as the fiber optic backbone and local drop capability of the common carriers has increased. It is anticipated that the cost reductions for T3 services will eventually follow that of T1 services.

SONET (OSI Level 1)

The next generation of synchronous transmission systems to be offered worldwide will be fiber optic based. SONET (the Synchronous Optical NETwork) network standards are now in place with widespread agreement between European and North American common carriers for deployment starting in 1991.

The SONET standards, much like the T-Carrier standards, provide a signal hierarchy (SDH - Synchronous Digital Hierarchy) of synchronous transmission services (STS), including STS-1 at 51.84 Mbits/sec, STS-3 at 155.52 Mbits/sec and STS-12 at 622.08 Mbits/sec. Other higher rates at least to STS-48, at 2.488 Gbits/sec, will eventually be provided.

These SONET services can be provided over either fiber or copper interfaces.

When provided over fiber these services are commonly referred to by their OC-n optical interface standard nomenclature. For example, OC-3, OC-12 and OC-48 refer to 155.52 Mbits/sec, 622.08 Mbits/sec and 2.488 Gbits/sec interfaces respectively.

In the United States, it is unlikely that STS-1 service will be widely available or used, if at all, as T3 is rapidly filling that speed niche. However, at rates above T3, e.g., STS-3c, SONET will be the primary common carrier service available.

Note that STS-Nc services mean that all the circuit bandwidth is delivered in a concatenated fashion, as opposed to an aggregation of the individual STS-1 units of 51.84 Mbits/sec being delivered over different paths, possibly at different times. This guarantees that a higher-speed application, such as a 2.4 Gbits/sec path over an STS-24c service, will see all of its bandwidth in proper sequence and time relationship.

B-ISDN and SMDS high speed data services will rely on SONET for deployment, as SONET becomes available (note that SMDS will first be delivered over DS1 and DS3 services).

Frame Relay (OSI Level 2)

Frame Relay is a new common carrier service that will provide a reasonable upgrade path from X.25 packet services. It appears that it will be offered in speeds up to 1.5 Mbits/sec. Frame relay is a virtual circuit based service, i.e., at least a semi-permanent virtual circuit must be established for each host-pair connection.

It seems clear that this offering has limitations for higher-speed backbone use, at least in its initial offerings. The target audience for Frame Relay does not appear to be higher performance networks; SMDS is slated for this.

SMDS (OSI Level 2)

Switched Multi-megabit Data Service (SMDS) is a recent common carrier offering developed by Bellcore[9]. Though initially based on the IEEE 802.6 MAN protocol, Bellcore emphasizes SMDS as a service, not a technology; i.e., they will introduce other access protocols for the user, as well as other long-haul protocols as appropriate in the future. Bellcore proposes supporting SMDS services over ATM when it becomes available.

SMDS uses cell based switching (to be compatible with ATM) and delivers packets up to 9188 bytes long for its level two function. SMDS is now just beginning

[9] Bell Communications Research (Bellcore) is the organization that was set up after the AT&T divestiture to provide research services to the regional Bell operating companies (RBOCs or "Baby Bells"). It plays the same role for the RBOCs that Bell Laboratories does for AT&T.

deployment, with several trials underway, including one centered at Temple University, and another centered at Stanford University.

ATM and B - ISDN (OSI Level 2)

B-ISDN, or Broadband-ISDN is a collection of services that will be provided via ATM (Asynchronous Transfer Mode) use of SONET networks, and will be the common-carrier wide-area high-performance network for the future. ATM is cell based, i.e., its atomic transmission unit is very small (48 bytes plus 5 byte header), and will offer datagram as well as virtual circuit services, ATM will carry digitized voice, video and data.

ATM will initially operate over SONET links at speeds of 150 Mbits/sec and 600 Mbits/sec, with the expectation of higher SONET rates in the longer term.

10. Acknowledgements

The authors would like to thank Van Jacobson of LBL for his ongoing support in providing access to systems running his latest TCP/IP protocol implementations; Wendy Huntoon, Jamshid Mahdavi, and Ralph Roskies of the Pittsburgh Supercomputer Center for their cooperation and collaboration with the case study; R. L. Fink of the LBL network and telecommunications group for providing much of the material in Appendix C; and Peter Schroder, formerly of Thinking Machines Corporation, Carl Crawford of General Electric Company, and Mark Roos of the LBL Research Medicine Division for their support and/or help with this project.

11. References

[Beguelin]

Beguelin, A., J. Dongarra, G. Geist, R. Manchek, and V. Sunderam, *Graphical Development Tools for Network-Based Concurrent Supercomputing*, Proceedings from Supercomputing '91 Conference Proceedings, pp. 435-444.

[Birns]

Birns, P., P. Brown, J. Muster, *UNIX for People*, (book), Prentice-Hall, 1984

[Catlett]

Catlett, C., *In Search of Gigabit Applications*, IEEE Communications Magazine, April 1992.

[Cline]

Cline, H., W. Lorensen, S. Ludke, C. Crawford, and B. Teeter, *Two Algorithms for the Three-Dimensional Reconstruction of Tomograms*, Medical Physics, May 1988.

[Comer]

Comer, D., *Internetworking with TCP/IP*, 2ed., (book), Prentice-Hall, 1991

[Corbin]

Corbin, J., *The Art of Distributed Applications: Programming Techniques for Remote Procedure Calls*, (book), Springer-Verlag, 1990

[Cysper]

Cysper, R., *Communications for Cooperating Systems: OSI, SNA, and TCP/IP*, (book), Addison-Wesley, 1991

[FCCSET]

Grand Challenges: High Performance Computing and Communications - The FY 1992 U. S. Research and Development Program, A Report by the Committee on Physical, Mathematical, and Engineering Sciences; Federal Coordinating Council for Science, Engineering, and Technology, and; Office of Science and Technology Policy

[Foley]

Foley, J., A. van Dam, S. Feiner, J. Highes, *Computer Graphics: Principles and Practice*, 2 Ed., (book), Addison-Wesley, 1990

[IEEE]

Gigabit Network Testbeds, IEEE Computer 23(9), September, 1990.

[IEEE-MSS]

Mass Storage System Reference Model: Version 4 (May, 1990), S. Coleman and S. Miller, Eds. (Developed by the IEEE Technical Committee on Mass Storage Systems and Technology.)

[Lee]

Lee, E., P. Chen, J. Hartman, A. Chervenak Drapeau, E. Miller, R. Katz, G. Gibson, D. Patterson, *RAID-II: A Scalable Storage Architecture for High-Bandwidth Network File Service*, Report No. UCB/CSD 92/672, February 1992; Computer Science Division (EECS), University of California, Berkeley; Berkeley, CA, 94720.

[Jacobson88]

Jacobson, V. and R. Braden, *Proposed Standard TCP extensions for long-delay paths*, ARPANet Working Group Requests for Comment RFC1072, DDN Network Information Center, Menlo Park, CA. October, 1988.

[Jacobson88a]
Jacobson, V., *Congestion Avoidance and Control*, Proceedings of the ACM SIGCOMM '88 Workshop, Stanford, CA., August 1988

[Jacobson90]
Jacobson, V., R. Braden and L. Zhang, *Proposed Standard TCP extension for high-speed paths*, ARPANet Working Group Requests for Comment RFC1185, DDN Network Information Center, Menlo Park, CA. October, 1990.

[Libes]
Libes, D., S. Ressler, *Life With Unix*, (book), Prentice-Hall, 1989

[Markoff]
Markoff, J., "*Creating a Giant Computer Highway-Robert Kahn's vision of a national network of information begins to take hold*, New York Times (Business Section), Sunday, Sept 2, 1990.

[Narten]
Narten, T., *Internet Routing*, Computer Communications Review, v19, n4, 1989 (Proceedings of the ACM SIGCOMM '89 Workshop, Sept. 1989)

[Rasure]
Rasure, J., C. Williams, *An Integrated Data Flow Visual Language and Software Development Environment*, Journal of Visual Languages and Computing, Vol. 2, No. 3, pp. 217-46, September, 1991. (For more information send e-mail to khoros-request@chama.eece.unm.edu)

[Schneider]
Schneider, M., *Pittsburgh's Not-So-Odd Couple*, Supercomputing Review, August, 1991.

[Stevens]
Stevens, W., *Unix Network Programming*, (book), Prentice-Hall, 1990

[Sun85]
The Unix System: A Sun Technical Report, Sun Microsystems, 1985 (part number 800-1419-02)

[Sequoia]
Sequoia 2000: A Multimedia Large Capacity Object Server, M. Stonebraker, Computer Science Division, University of California, Berkeley, and J. Dozier, Center for Remote Sensing and Environmental Optics, University of California,

Santa Barbara, Project Directors

[Sunderam]

Sunderam, V., *PVM: A framework for parallel distributed computing.* Concurrency: Practice and Experience, 2(4):315 - 339, December, 1990. (For more information send the following message by e - mail, to netlib@ornl.gov. Message: ''send index from pvm'' , or send e-mail to pvm@msr.epm.ornl.gov)

[Upson]

Upson, C., et. al., *The Application Visualization System: A Computational Environment for Scientific Visualization*, IEEE Computer Graphics and Applications, July, 1989, 30 - 42. (For more information contact Advanced Visualization Systems, Inc. at 617 - 890 - 4300.)