High Performance Communication in Large MTU Networks

Peter W. Haas, Rechenzentrum Universität Stuttgart (RUS), Stuttgart, Germany

ABSTRACT: *European Projects such as RACE 2031 PAGEIN – multimedia supported collaborative supercomputing, simulation and visualization in aerospace research and industry – are essentially dependent on local, national and international networking facilities. Previous observations of large MTU (maximum transmission unit) networks, like UltraNet and HIPPI, noted a number of shortcomings in the implementation of host protocol stacks and user applications. This paper will present a thorough analysis of network metrics in HIPPI local area networks that have been carried out in the context of PAGEIN. By inspecting actual source codes, a number of guidelines will be derived that yield both better understanding and substantial throughput improvements with most TCP/IP and UDP/IP based applications.*

1 Introduction

The University of Stuttgart Regional Computing Center (RUS) runs one of the major supercomputer-network complexes in Germany. The present RUS computer configuration comprises a Cray C-94/T-3D compute server, a combined Cray Y-MP file/tape archive server, an Intel Paragon massively parallel processor and a number of parallel workstation clusters. A large visualization laboratory deals with the rendering and animation aspects of comprehensive data sets that commonly evolve out of scientific/technical simulations. The networking infrastructure, necessary to interconnect all these systems, ranges from Ethernet, FDDI, and local ATM to the Gbit/s crossbar-switched High Performance Parallel Interface (HIPPI) network.

Experience with Gbit/s networks is available at RUS since the first operation of Ultra Network's UltraNet in 1989. Supercomputer protocol stack performance has been carefully analyzed during that time for all sorts of networks, and new networking technologies are constantly evaluated and compared against this data base. An overview of this work may be found in [1], [2], [3], [4], [5], [6], [7].

2 Typical Data Transfer Rates

In this chapter we want to present a tabular overview of typical user applications within an university computer center and their respective performance over various types of network media. The applications are separated into two classes according to bandwidth and datagram size requirements. As can be seen from **Table 1**, the first class mostly contains low-bandwidth services, like electronic mail, terminal emulation and shared file systems using the Network File System (NFS) protocol. The transfer rates given in **Table 1** are neither thrilling nor would they require the use of an exceptionally fast network, like HIPPI. With the exception of the NFS applications they even could be accomodated by a single 10-Mbit/s Ethernet. However, the last line clearly marks the transition point where the scope of the Fibre Distributed Data Interface (FDDI) will be left: Not so much due to a general lack of network bandwidth, but simply due to the dramatic reduction in the number of NFS transactions that comes along with the much larger datagram size available in a HIPPI network.

In the second class of applications, contained in **Table 2**, the HIPPI network constantly develops an order of magnitude increase in throughput over FDDI. We will see later on that this improvement is only partially due to the higher line rate of HIPPI. In the FTP and IPC case e.g., the major contribution comes from the adoption of 60-Kbyte datagrams in conjunction with a 360-Kbyte TCP flow control window. One is tempted to say that TCP flow control windows of similar size could be used over an FDDI network. However, in that case the TCP segment will be 4 Kbytes only, and on the order of 90 datagrams would be in flight for a single TCP connection. There is hardly any FDDI interface on the market that can deal with such an enormous amount of outstanding datagrams without resorting to hardware flow control on the physical level. Unfortunately, neither the Ethernet or next generation Fast Ethernet, nor the FDDI protocol have anything in mind with physical-level flow

Application	**Ethernet** [Kbyte/s]	**FDDI** [Kbyte/s]	**HIPPI** [Kbyte/s]
Electronic Mail	**0.01**	**0.01**	**0.01**
World Wide Web (WWW)	**3 - 10**	**3 - 10**	**3 - 10**
Terminal Emulation	**0.3 - 200**	**0.3 - 400**	**0.3 - 1,000**
ASCII Text Transfer	0.3	0.3	0.3
WYSIWYG-Text Processing	10 - 100	10 - 100	10 -100
Character Input	15	15	15
Scrolling in Documents	30	30	30
Rapid Mouse Movement	50	50	50
Archiving			
Tape Archiver (tar)	200	500	1.000
Network File System (NFS)			
Standard NFS (Read)	800	1,800	2,000
Standard NFS (Write)	100	400	3,000
NFS Version 3 / Cray NFS	800	2,000	8,000

Table 1: Typical Data Transfer Rates, Class 1

Application	**Ethernet** [Kbyte/s]	**FDDI** [Kbyte/s]	**HIPPI** 1) [Kbyte/s]
File Transfer (FTP)			
Binary Mode	800	3,000	35,000
ASCII-Mode	400	2,000	20,000
Distributed Mass Storage			
RAID-Systems using IPI-3	- - -	10,000	75,000
Interproc. Communication			
Memory-to-Memory (TCP/IP)	1,000	10,000	72,000
Moving Pictures, Television			
Video Codec, MPEG 1	200	200	- - -
Digital HDTV, MPEG 2	- - -	2,500 2)	2,500 2)
Framebuffer, 1280 x 1024 Pix	- - -	- - -	95,000

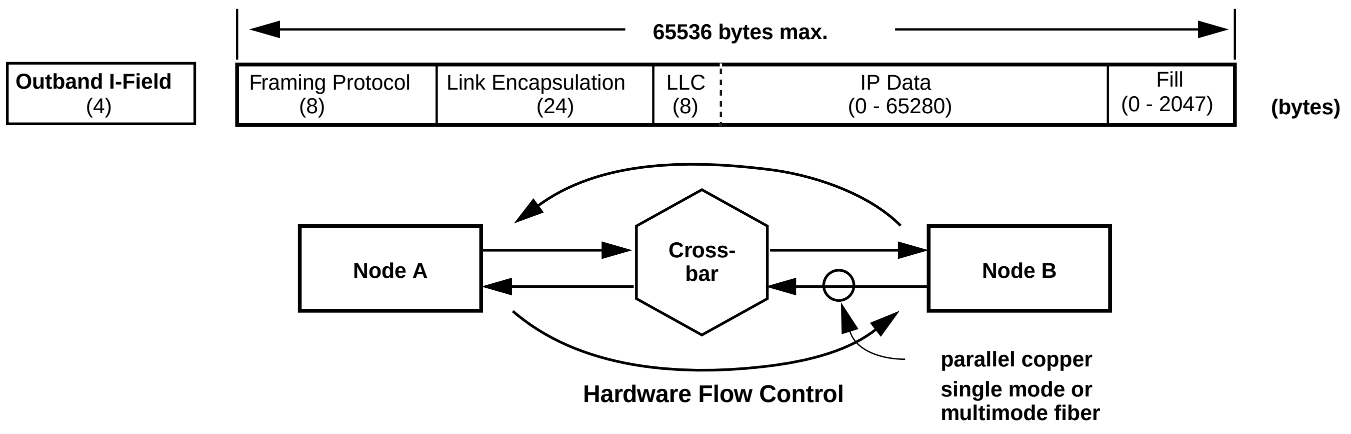
1) The usable data rate is 100 Mbyte/s or 200 Mbyte/s per host connection

2) estimated value

Table 2: Typical Data Transfer Rates, Class 2

control. Even worse, switching between multiple segments/rings in the latter case goes without any protocol support at all. It doesn't matter whether Mac-layer bridges or network-layer routers are used for switching. In both cases the switching device has to temporarily buffer the sum of all flow control windows that belong to the multiplicity of simultaneously active transport connections going across.

Looking at the amount of buffer storage that is usually available with either multiport bridges or IP routers, less than 10 Mbytes in any case, it becomes readily apparent why the HIPPI protocol tries to avoid storage within switching devices at all. Here the end systems themselves are responsible for the allocation and administration of buffer queues, if necessary, and an end-to-end hardware flow control scheme is used to prevent destination overflow. **Figure 1** summarizes the properties of a crossbar-switched HIPPI network. The dual-simplex HIPPI links are arbitrated and flow-controlled independently at 800 or 1600 Mbit/s, respectively. Although there is no physical limitation for the length of a HIPPI frame in general, fairness of access to a shared medium dictates that a common upper bound should be obeyed. With regard to IP traffic over HIPPI, the maximum frame length will be exactly 64 Kbytes which allows



Minimum Frame Length:	**8 bytes**
Maximum Frame Length:	**65536 bytes (LLC encapsulated IP, arbitrary length otherwise)**
Maximum Roundtrip Time:	**n * 500 ns + wave propagation time (n: number of HIPPI crossbars)**
Access Mechanism:	**Arbitrated End-to-End Connection Request/Indication**
Line Signal Encoding:	**plain 4-byte word with byte and burst parity, 20B24B-code for Serial_HIPPI**
Media Signaling Rate:	**800/1600 Mbit/s per simplex path**
Hardware Flow Control:	**64-Kbyte max Credits in 1-Kbyte increments, outband, end-to-end**
Switching Support:	**Outband I-Field with physical/logical addresses**

Figure 1: Characteristics of a crossbar switched HIPPI network

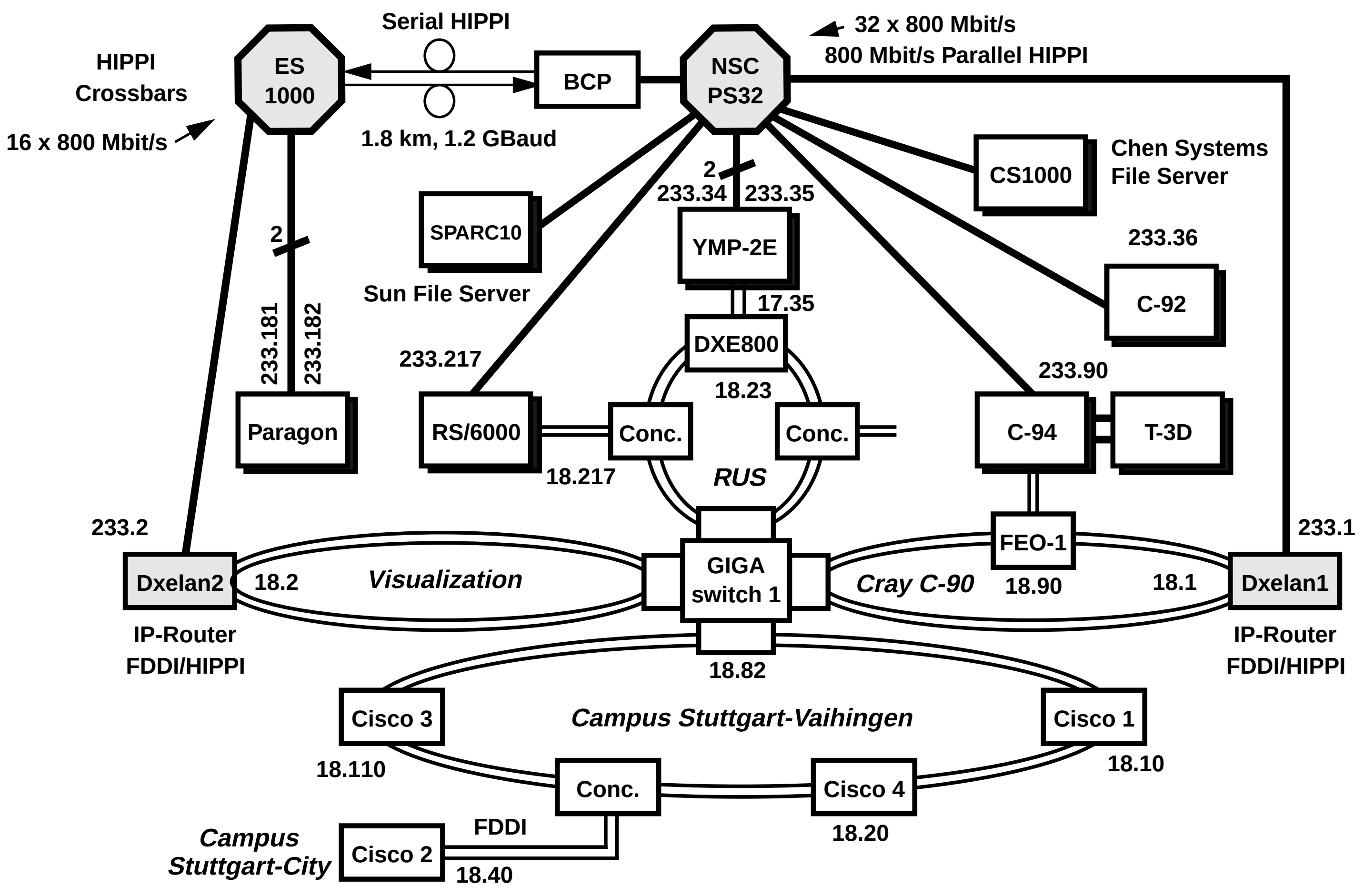


Figure 2: The RUS Computer and Network Configuration

for an IP payload of 65280 bytes, [8]. Therefore any individual IP host is supposed to release a HIPPI connection after no later than 660 µs. It should be noted that there is essentially zero storage along any of the (many) concurrent data paths going through a HIPPI crossbar. Therefore, once a point-to-point connection has been setup between Nodes A and B, the latency of a HIPPI switch will be effectively zero (500 ns in practice). It is also interesting to note that outband signalling will be used during the connection arbitration phase. This means that the functionalities of connection arbitration and data path cutthrough may be implemented in separate devices. Therefore, if an external arbiter is used, the switching of the data path may be accomplished by a pure photonic device, e.g. an all-optical crossbar, [9], or an optical star using wavelength division multiplex techniques, [10].

3 The RUS Network Configuration

The examination of network performance on the user application level, conducted in **Chapter 2,** will yield some basic understanding of the construction principles that have been observed in the topmost network configuration at RUS. Of course, a full representation of our network would not be very helpful in discussing the major design goals, therefore **Figure 2** has been simplified to a large extent. All low-speed and wide area networks have been left out. For the same reason we are not addressing our experimental ATM network, although it is quite large and handles a considerable amount of production traffic.

The university campus is covered by multiple FDDI rings which are switched on the MAC layer by doubly redundant GIGA crossbar switches (Digital Equipment). The campus itself is made up of two major sites, one located in downtown Stuttgart, the other one in the suburb of Vaihingen where most of the institutes reside. A total of twentyfour FDDI rings are needed to cover this area and the links in between. Whereas the university institutes access individual FDDI rings by means of FDDI concentrators or Cisco IP routers, that are fanning in and out mostly low-speed, low-MTU traffic, IP routing in a 24-ring FDDI backbone is not likely to work because of performance, cost and latency constraints. The DEC GIGAswitch has been found to offer an acceptable, though not perfect solution to switching. We use two of them in a balanced crossconnect topology, either one switching 12 out of 24 rings at full speed.

There is yet another backbone in **Figure 2** which is made up by the HIPPI network. Quite different from the FDDI complex, this backbone does not act as a turntable for transient traffic. Its main purpose is to provide a perfect, unimpeded networking environment for server machines of all kinds at the 800 Mbit/s level. More closely, perfect means:

- low latency (< 1 µs connection setup time on average)
- loss free (zero datagram drops)
- large MTU (65496 bytes max of IP data)

It is interesting to note the extremely low average value for the connection setup time of ~ 1 µs that has been observed with any of the compute servers (Cray C-90s, IBM RS/6000) or file server (Cray YMP). Of course, traffic entering the HIPPI network through any of the FDDI/HIPPI routers is not likely to add more than 10-percent of network load. So the chances for a HIPPI host to arbitrate a busy destination port, due to traffic originating from the FDDI network, are very low, indeed. However, this not so clear when destination blocking for local HIPPI traffic has to be considered. In this case, we both encounter high volume traffic and maximum sized datagrams at the same time. Obviously, for a zero-buffer switch the best solution is to increase the number of outgoing channels to severely loaded destinations. Most HIPPI switches know how to do load distribution over several parallel links to a common destination. This technique has been applied to the Cray YMP NFS server, the university's central data migration facility. In this case, the switch is authorized to use whatever link is currently available to the destination. It does so by choosing from a prespecified list of channels using various selection methods, e.g. priority, round robin or random access. With massively parallel machines (MPPs), like the Intel Paragon, the performance of a single network connection might not be adequate to meet the available traffic rates of other hosts in the network. Software-striping of outgoing parallel channels on the MPP is a very efficient solution. Again, on the reverse direction the switch state machine may be used to do the striping in hardware, so the peer application involved in the MPP connection does not need to know about striping at all.

UDP/IP fragmentation, needed for all versions of NFS, may be a big concern when the interworking between HIPPI and successively lower MTU networks, like FDDI and Ethernet, has to be considered. Although there is no specific rule where IP fragmentation should take place, the safest solution is to do it in the host that originates the transport connection rather than to leave it to subsequent IP routers in the network. This solution puts some extra burden on the hosts themselves, also path-dependent MTU values have to be administered, however, the big advantage is that the sequence of fragmented IP datagrams will be preserved under all circumstances. This is certainly not true for a Cisco router based IP network, where IP datagrams awaiting fragmentation may be delayed significantly over those that simply need to be forwarded. In general, host system processors will be at least as fast as (and cheaper than) protocol processors within dedicated routing equipment, so host fragmentation is the only way that is going to scale. We have found that most workstations, e.g. IBM RS/6000, SPARC10, may be favorably used for IP routing between HIPPI and lower speed networks.

4 TCP/IP, FTP and NFS Performance

The previous discussion has raised sufficient interest in the question what kind of variables are affecting protocol performance and how they do that in detail. It is wise to restrict ourselves to the investigation of very simple applications and to the optimization of the most dominant variables which the system programmer and eventually the user can actually influence. We will first concentrate on the TCP/IP memory to memory transfer which is almost entirely coded inside the operating system kernel. Here the major variables affecting performance may be summarized as follows:

1. Size of data copied from user space to kernel space
2. Number of data copies and checksum operations
3. Maximum transmission unit (MTU) of physical media
4. Size of kernel buffers for sending and receiving
5. Size of read()/write() data from user application
6. Size of TCP window scale option
7. Host system I/O architecture
8. Type of operating system

Not all of these variables, e.g. 2., 7., 8., may be addressed by the system administrator or the user programmmer. In practice, however, it suffices to deal with a small subset of variables that is outlined by the ordinal numbers 3., 4., 5. and 6.

4.1 Memory to Memory TCP/IP

In **Figure 3** we present three performance graphs that show the TCP/IP memory to memory throughput between two Crays as a function of the user buffer size, i.e. the amount of data the user is copying from the HIPPI source to the HIPPI destination. Starting with the lowest performance graph, we use the Unicos standard parameter settings: IP-MTU=65496 byte, socket buffer size of 64 Kbyte, and hence a TCP flow control window of 64 Kbyte max. The resulting throughput is almost constant at 20 Mbyte/s over the entire range of user buffers. Looking at the TCP peer to peer round trip time, RTT=3.1ms, we can't simpl y have more throughput than given by the socket buffer size divided by RTT. In addition, by exploring the peak for the 10-Kbyte user buffer it seems that the kernel could be much faster in copying the user data into the kernel's socket buffer, if only we had a bigger socket buffer.

In the next graph we do nothing else but increase the socket buffer by a factor of six. Any user on a Cray may optionally do this according to the default configuration of the netvar database. We see that the kernel copy bandwidth leapfrogs by a factor of ten, to 234 Mbyte/s, but still the sustained throughput over the HIPPI network is close to 20 Mbyte/s.

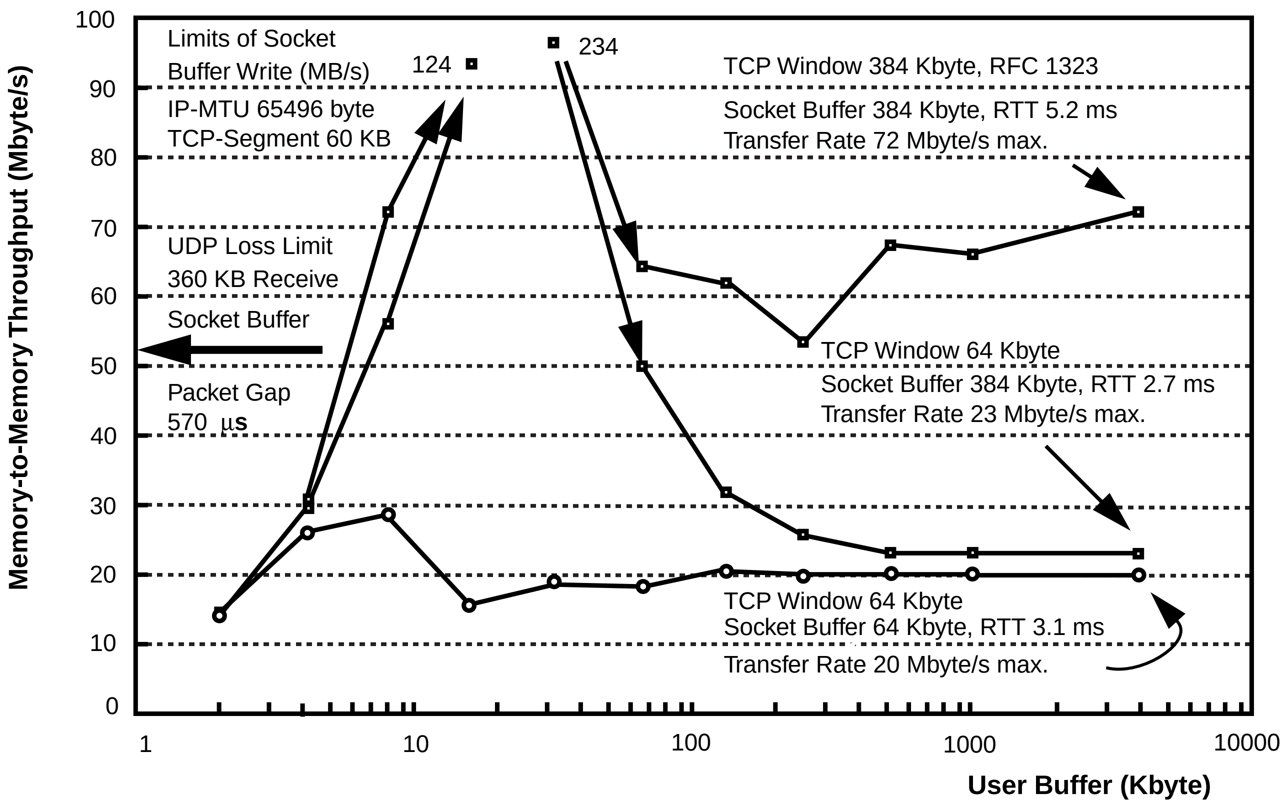


Figure 3: TCP/IP Memory-to-Memory Performance. Cray C-94 -> HIPPI Crossbar -> Cray YMP

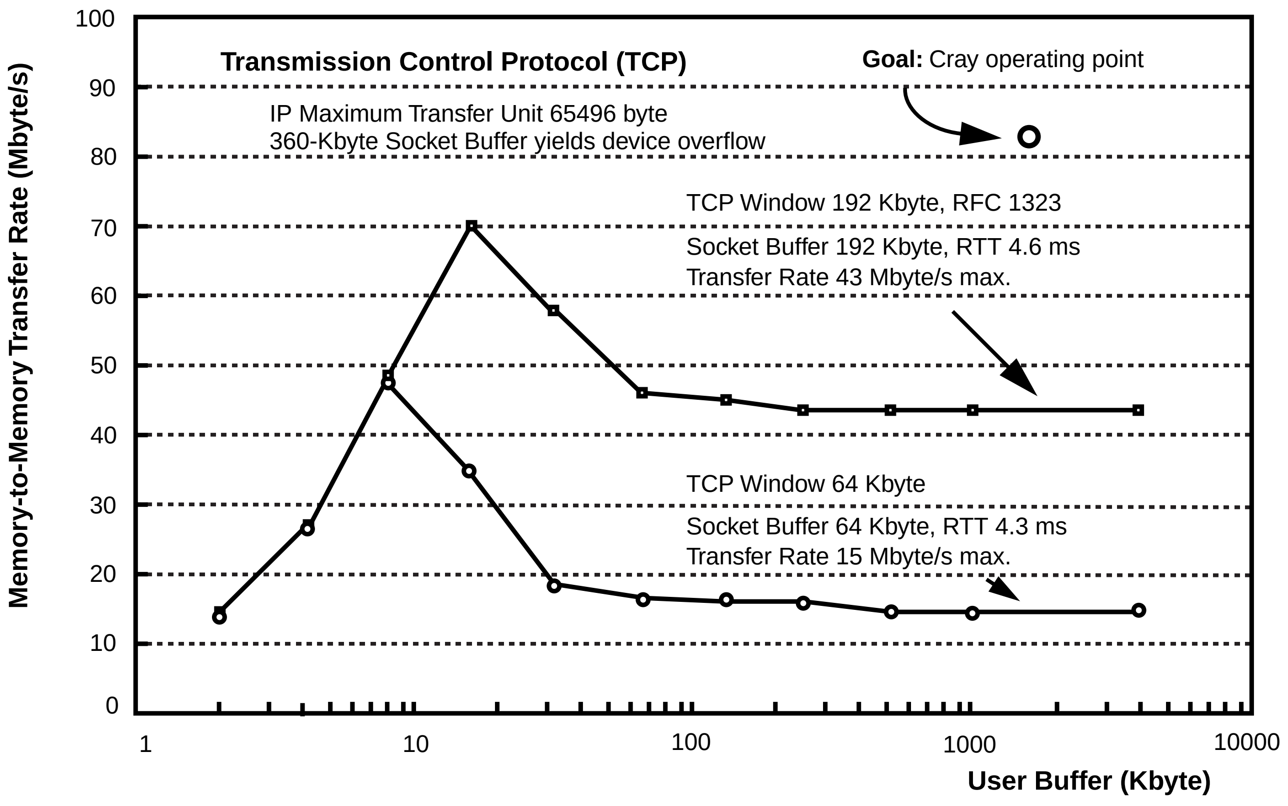


Figure 4: TCP/IP Memory-to-Memory Performance. IBM RS/6000 Model 590 -> HIPPI Crossbar -> Cray YMP

In order to increase sustained throughput we have to enlarge the TCP flow control window at least as much as the socket buffer in the second step. So, the final graph shows the memory to memory performance that comes along with both a scaled TCP window, according to RFC 1323, and a TCP socket buffer of 384 Kbyte. It is interesting to note that the full performance of approximately 70 Mbyte/s is reached over the entire spectrum of user buffer sizes, down to as low as 10 Kbytes. This ideal situation is likely to change if switches with internal buffering, i.e. non-zero latency, are employed.

Please keep in mind that the IP datagram size for all these measurements has been the TCP segment size plus the size of the combined TCP/IP header, roughly 60 Kbytes. Decreasing the TCP segment size below 60 Kbytes will considerably worsen any of the individual performance graphs, as indicated by the accompanying CPU utilization.

In **Figure 4** we give one example for a typical workstation performance over HIPPI. Although the discussion for the IBM RS/6000 would pretty much follow along the same lines as in the Cray case, the optimal parameter set might be different due to constraints at the HIPPI device level. It is essential to have a sophisticated socket test tool available, like tsock, that allows for the identification of any anomalies on the network and device level.

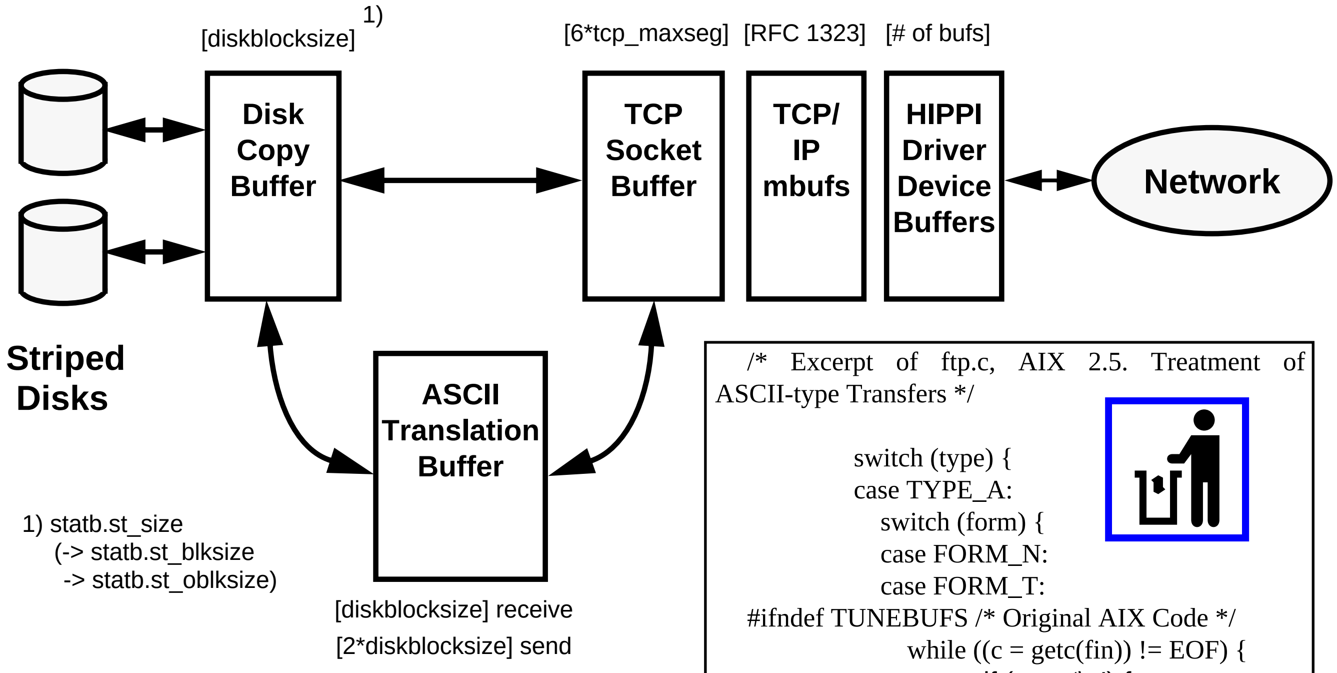
Although our measurements have been derived entirely from hosts connected to a HIPPI network, the basic assumptions should also be valid for other large MTU networks that operate in the same speed range, like the full-speed Fibre Channel Standard (FCS) and IP over ATM using AAL5 at OC-12c speeds. However, the connectionless FCS services (other than class 1) and IP over ATM in general have a non-zero loss probability that varies largely with network load and also IP datagram size.

4.2 Workstation File Transfer Protocol

The Unicos FTP model takes a number of intermediate buffers into account that are deliberately placed between the file and network I/O system as shown in **Figure 5**. These buffers are used for data streaming and consequently have to be (automatically) matched to the available disk and network block sizes.

The disk copy buffer is available with most FTP implementations and is generally of an arbitrary, fixed size, e.g. BUFSIZ, because the block size of the underlying physical disk(s) is mostly unknown. Unicos has some recommendations towards optimal disk block sizes for small and large transfers, based on the file's stat{} structure. The network copy buffer, so to speak, is the well known TCP socket buffer and its size has been sufficiently well investigated in **Chapter 4.1** alltogether with the need for the TCP window scale option. What's new is the ASCII translation buffer because block-processing character translations with optimized routines, like memccpy(), is much more efficient than using the GETC and PUTC character macros.

The Unicos FTP model described above has been around for some time, so it is supposed to be rather mature. However, it would be interesting to see how ordinary workstation FTP



```
    /* Excerpt of ftp.c, AIX 2.5. Treatment of
ASCII-type Transfers */

                switch (type) {
                case TYPE_A:
                    switch (form) {
                    case FORM_N:
                    case FORM_T:
    #ifndef TUNEBUFS /* Original AIX Code */
                        while ((c = getc(fin)) != EOF) {
                                if (c == '\n') {
                                    DOPUTC('\r');
                                }
                                DOPUTC(c);
```

Figure 5: Unicos Client File Transfer Parameters

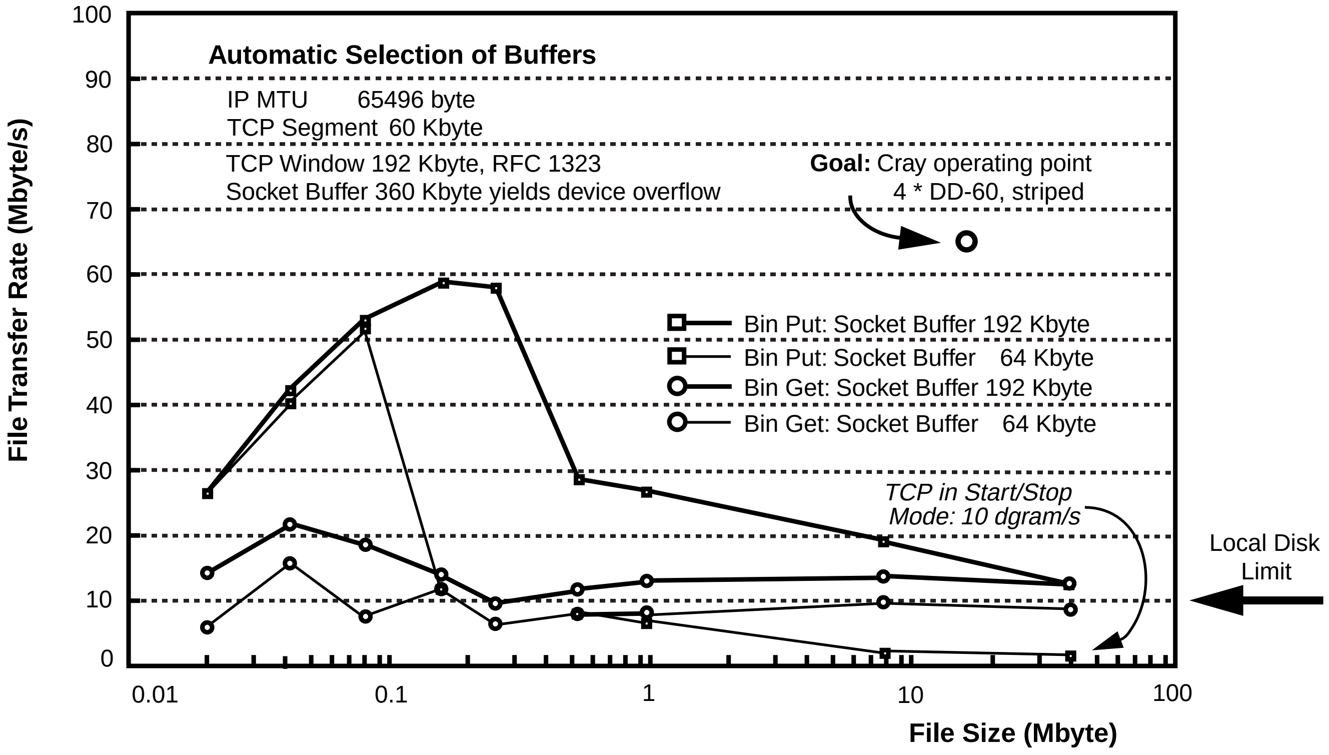


Figure 6: Workstation File Transfer Performance. IBM RS/6000 Model 590 -> Cray YMP

implementations behave in a large MTU environment and how they could benefit from the Unicos FTP version.

Figure 6 contains FTP performance graphs that have been gathered from a heavily loaded IBM RS/6000 batch machine. The topmost graph shows the binary put according to the Unicos model; there is an excellent match between the file system read ahead and the socket buffer write for file sizes up to 1 Mbyte. Actually, the FTP transfer seems to proceed faster than the disk can be read physically. With the AIX operating system there is no recommendation of an optimal disk block size. A first approach would be to match the disk copy buffer to the TCP socket buffer which in turn is six times the TCP maximum segment size, both peers had settled upon during connection setup. With an IP-MTU close to 64 Kbyte, TCP_MAXSEG tends to be negotiated down to 60 Kbyte, the next lower 4-Kbyte boundary. Looking more closely into the TCP performance chart of Figure 4, we decide to clamp both the socket and disk copy buffers at a maximum of 192 Kbytes.

Reducing the TCP socket buffer to 64 Kbytes with the binary put, reveals a very common anomaly in a network where the TCP window on the receiver may be fully closed by just a single incoming datagram. What happens is that the TCP transfer resorts to start/stop-mode: the flow control window oscillates completely between both extremes, and the resulting network throughput will be fairly constant, approximately 10 maximum sized IP datagrams/s. Fortunately, the AIX version of FTP uses smaller buffers by default and will not enter this operating region by accident.

The last two performance graphs show the binary get, again for 192- and 64-Kbyte socket buffers, respectively. The receive rates are still close to or slightly above the disk's physical write limit. In general, tuning the stream buffers according to **Figure 5** will yield a 50 to 100 percent improvement in a HIPPI network. In addition, a dedicated character translation buffer combined with highly efficient character matching routines will speed up ASCII-type transfers by an order of magnitude.

4.3 Cray Version of NFS

We have recently analyzed the Unicos 8.0 NFS implementation in order to maximize NFS performance over HIPPI and to prepare the proving grounds for various NFS version 3 beta tests, [6].

Figure 7 contains a simplified representation of the system buffers involved along the client's NFS write path to the server. The physical network is represented by a crossbarswitched HIPPI LAN. Under the Unicos 8.0 operating system, servers need user-level contexts to service multiple NFS requests in parallel. Daemons (nfsds, for Standard NFS, and cnfsds, for Cray NFS) provide these contexts for the server's kernel to use. On the client side, block I/O daemons (biods) perform asynchronous I/O.

A simple UDP/IP constant rate transfer, accomplished by the tsock test tool, will easily identify the UDP loss limit on Crays. According to the marker in **Figure 3**, it is slightly above 50 Mbyte/s on non-dedicated systems. We have extensively measured UDP datagram loss along the NFS write path during production time. Since the HIPPI network is essentially free of

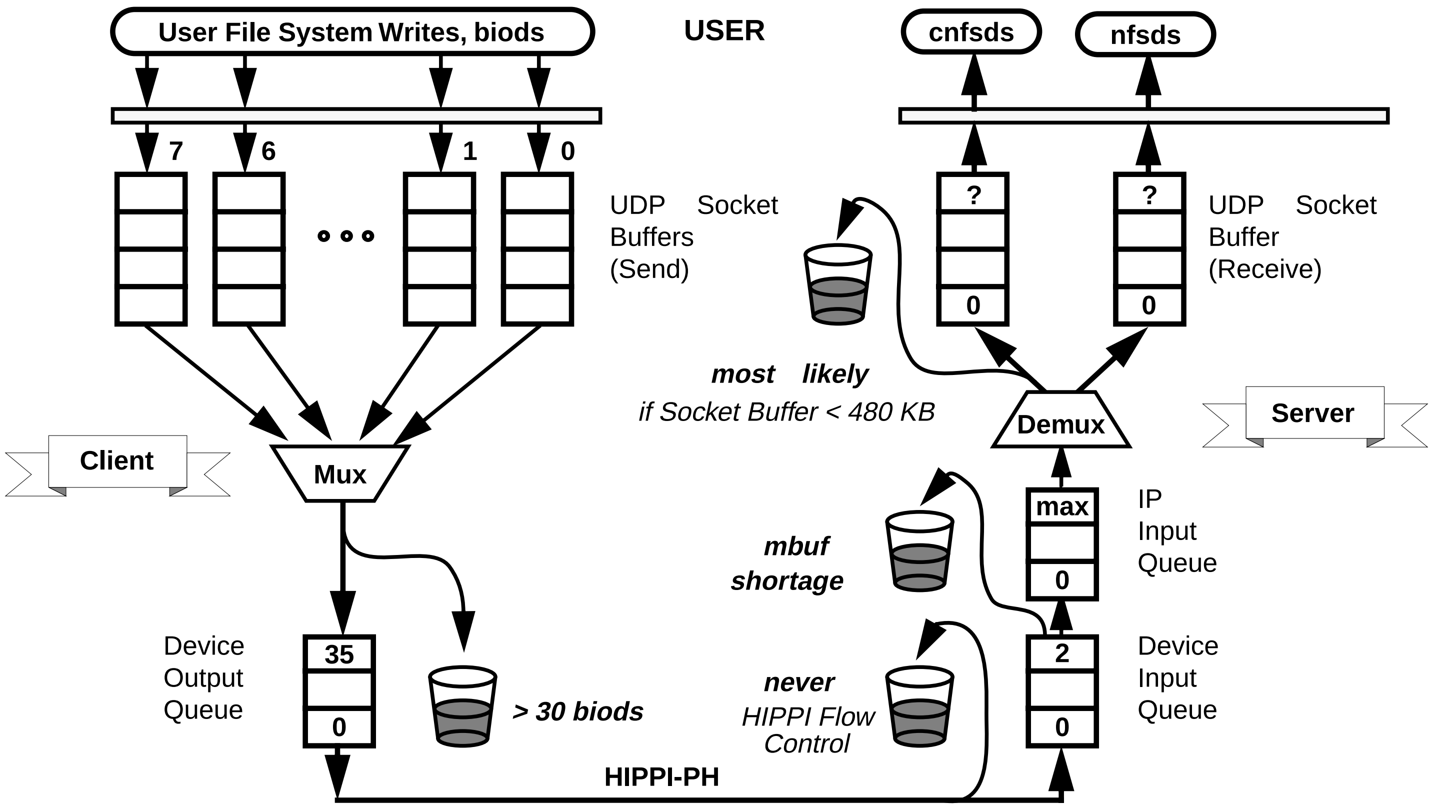


Figure 7: Datagram Loss with NFS Traffic

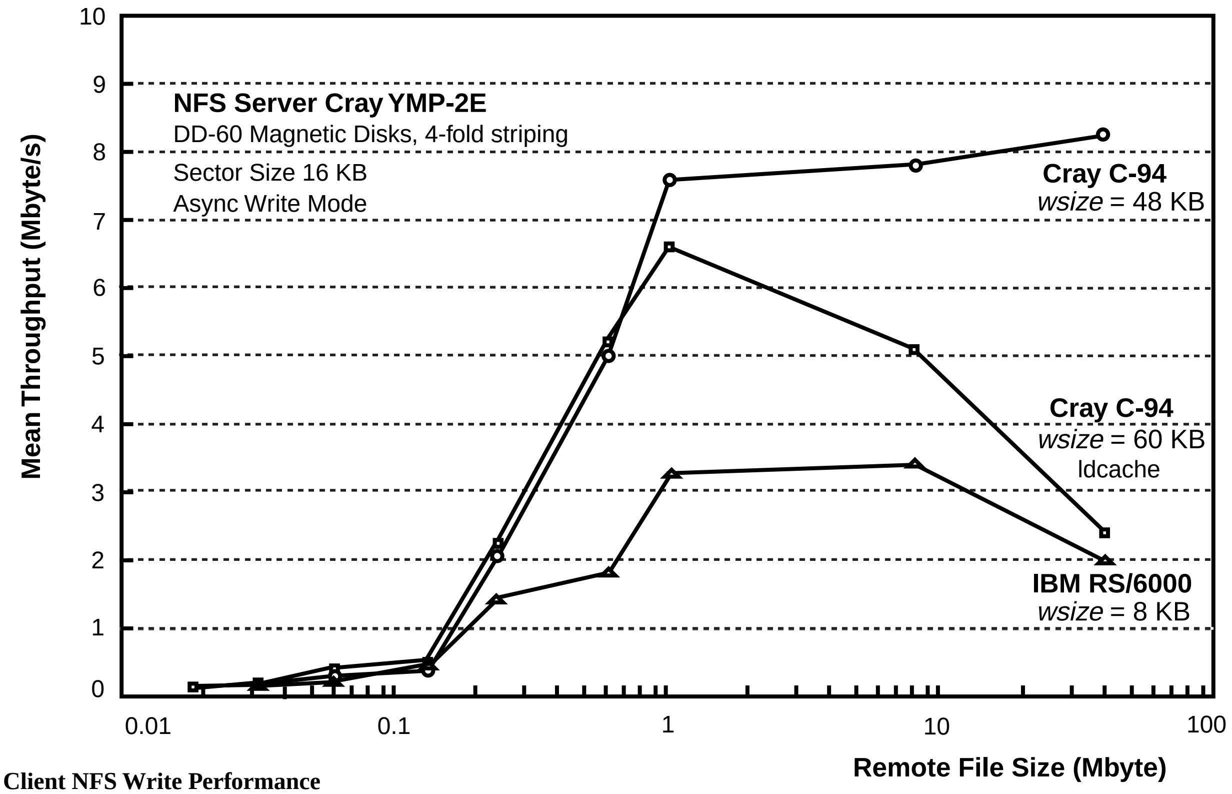


Figure 8: Client NFS Write Performance

loss due to Camp-on and physical hardware flow control, the only locations where datagram loss may occur are the client's device output queue, and the server's IP input queue and UDP receive buffer. Fortunately, if there is any UDP datagram loss on Crays associated with NFS traffic, it will almost always occur on input to the server's UDP receive queue. Of course, this case may be easily handled by increasing the socket buffer receive space appropriately.

It has been common experience for some time that NFS write requests do perform significantly less than read requests, especially, when synchronous writes to the remote file system are enforced. NFS writes may be done asynchronously under Unicos 8.0, much the same way as in NFS Version 3. **Figure 8** shows the average NFS performance associated with single asynchronous NFS writes. We see that NFS throughput on writes increases rather dramatically with write request size. Also, compared to reads, the rising edge in the performance plots has been shifted towards smaller file sizes because there is less processing overhead with a client write transaction. Comparable NFS performance has been measured with NFS version 3 in ATM networks recently, [11].

However, much like with ordinary TCP/IP the NFS flow control window of 64 Kbytes is now exhausted with a single UDP/IP datagram. The average read/write service times on our NFS server are about 7 ms including the buffered disk operation. For a 60-Kbyte read/write transaction this yields a theoretical throughput of 8.8 Mbyte/s max. at 140 transactions/s. A four-processor server is supposed to handle 600 transactions/s or even more, therefore, a much larger NFS flow control window is needed to exploit more NFS bandwidth (up to 50 Mbyte/s) for a single NFS transfer.

5 Distributed Visualization Pipelines

There is a new set of user applications that are heavily depending on networks, although most users don't know. The Collaborative Visualization Environment, COVISE, of the European Union PAGEIN project will be used as an example, [12].

The COVISE system architecture was designed to meet the requirements of a distributed environment for scientific simulation and visualization in terms of collaborative working facilities with integrated multimedia support, data base like object management, remote steering of simulations and efficient use of highspeed networks. COVISE puts its emphasis on high performance computing with efficient data handling between distributed modules. The available hardware resources such as file servers, vectorizing or parallel computers, graphics workstations are transparently integrated into a single software environment allowing scientists to concentrate on computational issues instead of dealing, e.g. with the questions of how to organize data transfers and lockstep communicating processes on remote machines.

We will briefly look at parts of the software architecture that are needed to simulate and render a vortex breakdown using the Pegase CFD package from ONERA, the computing center of the French aerospace industry. **Figure 9** contains a black and white representation illustrating one time step of the vortex breakdown, showing an ISO surface of the flow field with two interactively controlled cutting planes next to it.

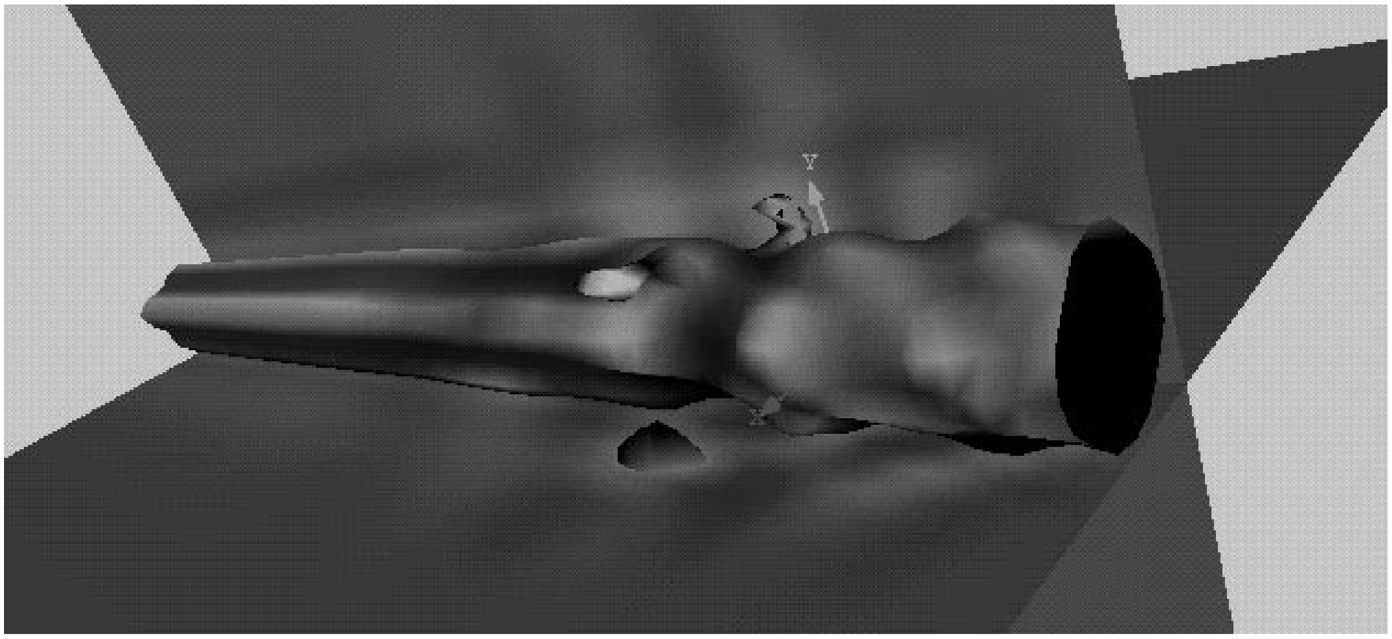


Figure 9: Simulation of a Vortex Breakdown by solving the 3D Navier Stokes Equations for an unsteady incompressible flow. Computation performed using Pegase from ONERA.

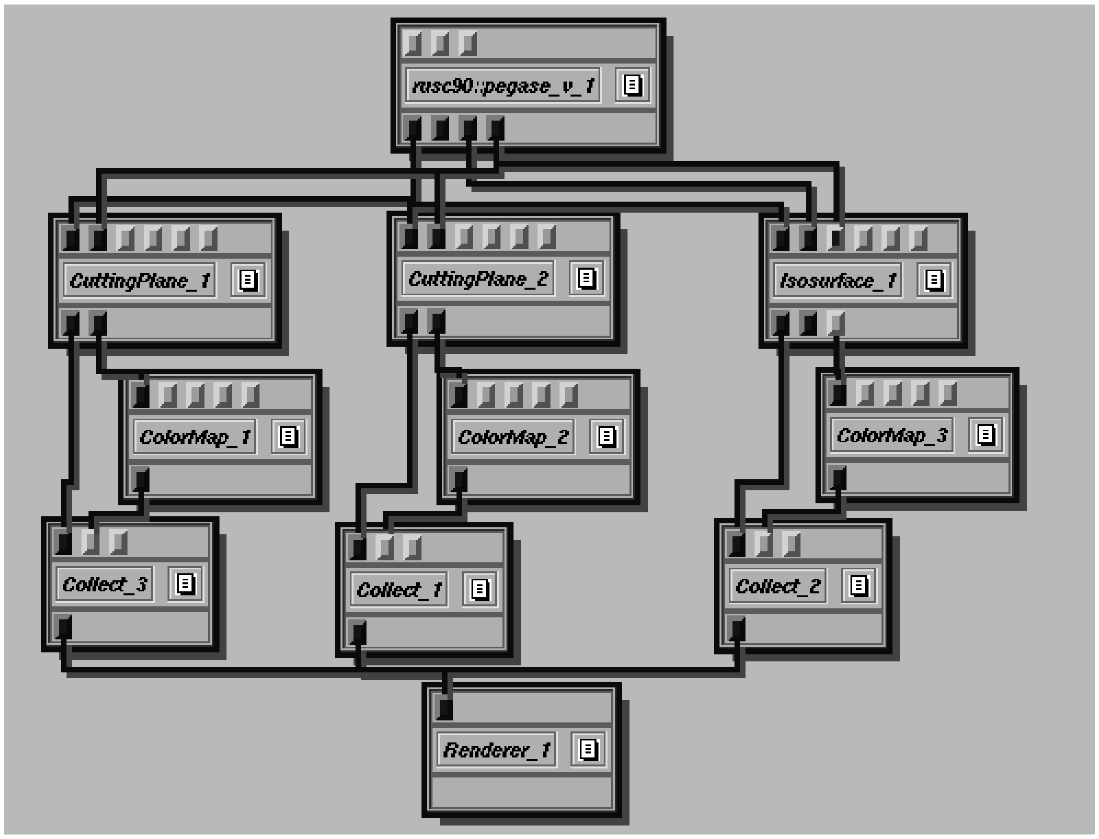


Figure 10: The COVISE visual command interface. Detail showing the MapEditor displaying Modules of a visualization pipeline.

Most currently available visualization systems follow the visual programming paradigm. This means that a certain visualization task is partitioned into subtasks each of which has input and output data. The complete visualization pipeline is built by choosing modules, each representing a subtask, and by interconnecting their respective input and output ports.

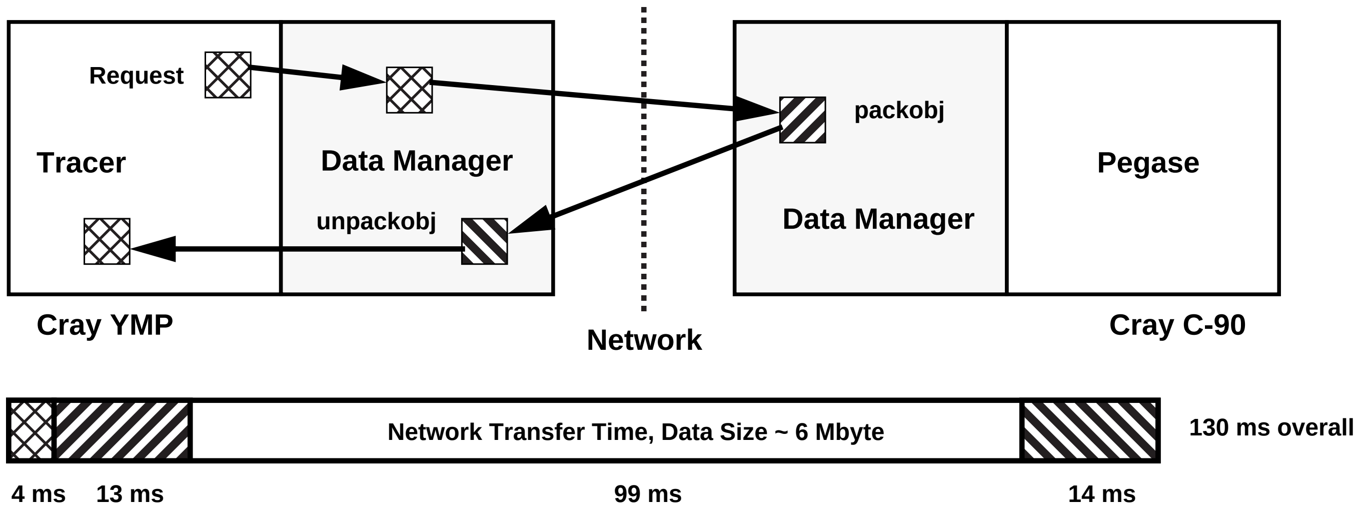
Figure 10 shows part of the COVISE visual command interface, the so-called MapEditor, displaying modules belonging to the visualization pipeline that has been composed for the simulation of the vortex breakdown. In the COVISE system a subtask is represented by a separate process. The system is designed in a way that the user can easily start processes on different computers. Every module is started on the computer where it runs best, and the input and output ports are automatically interconnected. In **Figure 10**, e.g. the Pegase code is running on the Cray C-90 whereas all other modules are kept on the same machine where the COVISE control session takes place.

In order to demonstrate the importance of network performance, **Figure 11** has a look at the individual stages of a data request that originates within the Tracer process on the Cray YMP. The distributed data manager takes care of the shared data space and the remote transfer of objects. In this case a remote data object of 6 Mbyte size, actually the contents of one cutting plane, is requested from the Pegase CFD process on the Cray C-90.

The overall processing time for the Tracer's data request is a mere 130 ms, translating into an application level throughput of 49.5 Mbyte/s, best case. According to the topmost performance graph in **Figure 3**, COVISE is adopting a 360-Kbyte TCP socket buffer and TCP window scaling across the HIPPI network. Of course, the responsiveness of large server machines varies drastically with workload. The average long term application level throughput therefore is only 12.5 Mbyte/s, with the worst case being even below 1 Mbyte/s, if any of the communicating processes is swapped out to disk temporarily. However, even for the best case the network takes up to 75 percent of the overall data request time, although running at 55 percent of the available physical bandwidth, [13].

Conclusion

We have demonstrated that vector super computers are close to Gbit/s speeds in the production environment if a number of network parameters are met. Most important are the datagram size and the amount of user data that may be in transit over the network. Workstation processors running at 100 million instructions per second close in on super computer network performance, but even here the network throughput scales



Throughput	best case	average	worst case
Application	49.5 Mbyte/s	12.5 Mbyte/s	0.25 Mbyte/s
Network	55.5 Mbyte/s	16.4 Mbyte/s	0.25 Mbyte/s

Figure 11: Individual Stages and Performance of a COVISE remote Data Request

almost linearly with datagram size. Network switches should not deal with complex protocols in the local environment, because they are not likely to keep up with the fast moving technology of end system processors. There is an increased interest in very low latency switches that provide a viable migration path towards protocol-transparent, all-optical networks in the future.

Glossary

ASCII	American Standard Code for Information Interch.
ATM	Asynchronous Transfer Mode
BCP	Broadband Communication Products
BSD	Berkeley System Distribution
CFD	Computational Fluid Dynamics
CNFS	Cray Network File System
CPU	Central Processing Unit
DEC	Digital Equipment Corporation
DXE	Data Exchange
ES	Essential Communications Switch
FDDI	Fibre Distributed Data Interface
IBM	International Business Machines
IP	Internet Protocol
ISO	International Standards Organization
FTP	File Transfer Protocol
HIPPI	High Performance Parallel Interface
IPC	Inter Process Communication
LAN	Local Area Network
LLC	Logical Link Control
MTU	Maximum Transmission Unit
NFS	Network File System
RPC	Remote Procedure Call
RUS	Regional Computing Center, U. of Stuttgart
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
OC	Optical Circuit
PAGEIN	Pilot Applications for a Gbit European Integrated Network
XDR	External Data Representation

References

[1] Haas, P., Christ, P., Rühle, R.: “User – The Stuttgart MAN Field Trial”, SMDS Conference, Berlin, Germany, October 12–13, 1992.
[2] Haas, P.: “UltraNet Release 4: Architecture and Performance Analysis”, Cray User Group (CUG) 1993 Fall Proceedings, pp. 185–212, Kyoto, Japan, September 20–23, 1993.
[3] Haas, P., Christ, P.: “Networking Issues in PAGEIN: The N of HPCN”, Lecture Notes in Computer Science, Volume 37, High-Performance Computing and Networking, pp. 86–93, Springer-Verlag, Berlin.
[4] Wierse, A., Lang, U., Nebel, H., Rantzau, D.: “The Performance of a Distributed Visualization System”, Proceedings of the International Workshop on Visualization, Paderborn, Germany, January 18–20, 1994.

[5] Haas, P.: “Classical Host Interfaces and Network Architectures Aiming at the Gigabit”, Networld+Interop 94, Berlin, Germany, June 8–10, 1994.

[6] Haas, P.: “Optimal UDP Buffering for Unicos 8.0 NFS”, Cray User Group (CUG) 1994 Fall Proceedings, pp. 197–204, Tours, France, October 10–14, 1994.

[7] Haas, P.: “High Speed Networking at RUS”, International Conference on High Performance Computing and Networking, Milan, Italy, May 2–5, 1995.

[8] Renwick, J., Nicholson, A.: “IP and ARP over HIPPI”, RFC 1374, Cray Research, Inc., October 1992.

[9] Optivision: “8x8 Optical Crossbar”, Optivision, Palo Alto, CA.

[10] Hall, E., et al.: “The Rainbow-II Gigabit Optical Network”, IBM T.J. Watson Research Center, Hawthorne, NY.

[11] Stuart, D.: “The Influence of ATM on Data Serving for High Performance Computing”, Maximum Strategy, Inc., Milpitas, CA, 1995.

[12] Rantzau, D. et al.: “Collaborative and Interactive Visualization in a Distributed High Performance Software Environment”, Proceedings HPCGV, pp. 194–203, High Performance Virtual Environments V, Swansea, UK, July 3–4, 1995.

[13] Wierse, A.: “Performance of the COVISE visualization system under different conditions”, Proceedings SPIE 2410, pp. 218–229, Visual Data Exploration and Analysis II, Grinstein and Erbacher, Ed., 1995.