October 1985

Report No. STAN-CS-86-1117
Also numbered CSL-85-287

An Empirical Study of Distributed Application Performance

by

Keith A. Lantz, William 1. Nowicki, and Marvin M. Theimer

'Department of Computer Science

Stanford Univcrsitp
Stanford, CA 94305

An Empirical Study of Distributed Application Performance

Keith A. Lantz, William I. Nowicki, and Marvin M. Theimer

Abstract

A major reason for the rarity of distributed applications, despite the proliferation of networks, is the sensitivity of their performance to various aspects of the network environment. We demonstrate that distributed applications can run faster than local ones, using common hardware. We also show that the primary factors affecting performance are, in approximate order of importance: speed of the user's workstation, speed of the remote host (if any), and the high-level (above the transport level) protocols used. In particular, the use of batching, pipelining, and structure in high-level protocols reduces the degradation often experienced between different bandwidth networks. Less significant, but still noticeable improvements result from proper design and implementation of the underlying transport protocols. Ultimately, with proper application of these techniques, network bandwidth is rendered virtually insignificant.

This research was supported by the Defense Advanced Research Projects Agency under contracts MDA903-80-C-0102 and N00039-83-K-0431, by the National Aeronautics and Space Administration under contract NAGW-419, and by the National Science Foundation under a Graduate Fellowship. This paper was published in *IEEE* Transactions on *Software Engineering SE-1* 1(10):1162-I 174, October 1985.

1. Introduction

Despite the proliferation of computer networks, distributed application programs are still uncommon. A major reason for this is the sensitivity of these applications' performance to various aspects of the network environment in which they are run. In addition to the inherent cost of the computation, the cost of communication between the distributed parts of the application are incurred. Consequently, the total computation cost of a distributed program is almost always higher than the total computation cost of an equivalent centralized program.

There are two approaches to improving the performance of distributed applications. The traditional approach is to improve the performance of the underlying network communication mechanism, possibly with problem-oriented protocols [20]. Another approach is to decrease the amount of network traffic by judicious partitioning of responsibility between the distributed components of the application, together with high-level protocols that reduce the frequency and volume of communication and that allow concurrent operation of the various components.

For comparison, consider the many performance studies made of demand-paged virtual memory systems. Although performance can be improved by speeding up the handling of page faults, better results are usually achieved by reducing the number of page faults. For example, increasing physical memory, tuning the page size, improving the locality of the application, or using a better replacement algorithm can make as substantial a difference as buying a faster disk.

One method of distribution that is. becoming increasingly popular, especially where graphics is concerned, entails the use of *backend* computing engines which communicate with *frontend* display facilities. This organization seeks to split out user-interactive functions that are especially sensitive to time delays into the frontend, leaving functions that are less time critical in the backend. However, such a separation may still involve fairly extensive interaction between the various parts of the distributed system, implying that care must be taken to avoid having the network communication become a bottleneck.

The V distributed operating system (V-System) being developed at Stanford supports distributed graphics applications of the sort just described [3, 9, 23]. This paper describes experience gained with the V-System with respect to various factors that affected those applications, performance. Section 2 describes the V-System environment. Section 3 describes the observed performance behavior of distributed applications. The six subsequent sections analyze various factors in some detail, including the effects of processor speed (Section 4), issues in high-level protocol design (Section 5), general issues in transport protocol design (Section 6), and a detailed discussion of one particular transport protocol, namely, ARPA Internet TCP [18] (Section 7). Conclusions are drawn in section 10.

2. Overview of the V-System

The V-System is a message-based distributed operating system designed primarily for high-performance workstations connected by local networks. It permits the workstation to be treated as multi-function component of a distributed system, rather than solely as a intelligent terminal or personal computer. Ultimately, it is intended to provide a general-purpose program execution environment similar to some degree to UNIX [19].

2.1. Hardware Environment

The V-System is being developed within the hardware environment of the Stanford University Network (SUNet). SUNet is a rapidly evolving environment consisting of:

- workstations, such as the Dolphin, Lisp Machine, Sun [1] and IRIS [6];
- standard timesharing systems, such as DecSystem-20/TOPS-20, VAX/UNIX, and

VAX/VMS; and
- dedicated server machines, for printing, file storage, and gateway services;

interconnected by various local networks, including 3 and 10 Mbit Ethernet [14]. Various machines are also connected to long-haul networks such as the ARPANET.

SUNet is representative of the workstation-based distributed systems currently in place or being developed at many locations worldwide. As a result, the V-System architecture is well-suited to any such system.

2.2. Software Architecture

The V-System consists of a distributed kernel and a distributed set of server processes. The distributed kernel provides network-transparent interprocess communication based on *synchronous* message-passing — such that a sender blocks until a reply is received. It consists of the collection of kernels resident on the participating machines. The host kernels are integrated via a low-overhead *inter-kernel* protocol (IKP) that supports transparent interprocess communication between machines [3].

Servers include device servers, storage servers, virtual graphics terminal servers, exception servers, and network servers. The following section discusses network services in some detail.

Lastly, a standard program environment has been defined, the principal instance of which is a C program library. The C library includes runtime support for standard C and UNIX-like library functions to facilitate the porting of existing C programs.

2.3. Network Services

The V-System supports network transport through three different protocol families. The standard means of communication between hosts on the same local network is by means of the inter-kernel protocol (IKP) mentioned above. The typical use of IKP is as a a reliable datagram service, such that packets are delivered, but duplicates are not suppressed.' IKP is implemented directly on top of the data-link level and is used by each host's kernel whenever interprocess communication is requested with a non-local process. Reliable byte-streams (henceforth referred to as V I/O connections) are provided external to the kernel by means of a reliable block I/O protocol that suppresses duplicates, together with a library package that provides a byte-stream interface to that block I/O protocol [23].

Access to hosts that do not reside on the same local network or that do not support the V-System's specialized communications protocols is provided by means of the Xerox PUP [2] and ARPA Internet [17] protocol families. The two families are supported up through the network and transport levels, respectively, in the form of an *internet server.* Higher-level protocols, such as TELNET for remote terminal access, are provided as separate packages that interface to the internet server via V I/O connections.

Since it is envisioned as the primary interface to hosts beyond the local network, the internet server has been designed to allow easy addition of other protocol families. That is, it has been structured with an eye towards flexibility at the expense of some speed. Clients of the internet server interact with it by means of V I/O connections. The server interfaces these connections to whatever network protocol has been requested. Thus, clients rarely need concern themselves with the details of the network protocols used to access a remote host. Protocol concerns only appear when creating a connection instance and when protocol-specific commands need to be specified to the server.

'In fact, clients may request that duplicates be suppressed, but this feature is rarely used.

2.4. Application Model

From the previous discussion it should be apparent that applications may run local to the user's workstation or on any other host accessible via the various communications protocols. Ultimately, all applications must communicate with the user via the virtual graphics terminal *server* (VGTS) resident on the user's workstation [9,16]. The VGTS provides the usual facilities present in contemporary window systems, including the ability to run any number of applications simultaneously, mapping them to the display when and where the user desires.

However, the VGTS is distinguished from most other window systems by two key features. First, it is designed to operate in a environment composed of a variety of applications, machines, and networks, with widely varying terminal interaction requirements. In contrast, most window systems have confined themselves to homogeneous environments, which require less flexibility in the window system.

Second, the VGTS supports structured graphics. Specifically, a graphical object can be defined in terms of other objects, which can in turn be defined in terms of yet other objects. Thus, the VGTS supports *structured display files* rather than the more common *segmented display files* [15]. The resulting *virtual graphics terminal protocol* (VGTP) is a high-level object-oriented protocol that attempts to limit both the frequency of communication between application and VGTS and the amount of data transmitted at any one time.

The VGTP is constant over all applications. However, some applications have no knowledge of the VGTP and some applications are running on machines that do not support the interprocess communication mechanisms underlying the VGTP. The following situations arise (see Figure 2-1, where each inter-machine arc is labeled with an example *(presentation protocol, transport protocol)* pair):

- Application A runs on the workstation and communicates via the VGTP. Current examples include text editors, document illustrators, and VLSI design aids.
- Application ***B*** runs on a machine that supports V kernel services, specifically, network-transparent interprocess communication via IKP. ***B*** communicates with the VGTS via the VGTP, as in the case of a application A.
- Application C runs on a machine that does not support IKP, but does support a traditional network architecture such as the Internet protocol family [17]. In addition, a VGTP interface package is available that encapsulates the VGTP within the appropriate transport protocol. Similarly, a local *agent* for the application, C', is created on the workstation to decapsulate the VGTP. Thus, the application may still be written in terms of the VGTP and neither it nor the VGTS have any knowledge that the other is remote. Our VLSI layout editor, for example, can be run in this fashion under VAX/UNIX.
- Application ***D*** has no knowledge of the VGTS or the VGTP; it wishes to regard the workstation as just another terminal. The local agent, ***D'***, is "user TELNET" and performs the appropriate translations between TELNET and VGTP.
- Application ***E*** is distributed between the workstation and one or more other machines. The local agent, ***E'***, is responsible for representing the multitude to the VGTS. It must perform the appropriate set of protocol conversions indicated above. In addition, it may wish to perform application-specific functions, such as high-level caching. In that case, the protocol used to communicate with the remote applications may require more than simple transport service.

All applications but *A* make use of a network transport protocol, whether they realize it or not. Application ***B*** employs an interprocess communication protocol that has nothing to do with graphics *per se.* Application ***D*** employs a protocol that in no way depends on knowledge of the VGTS and typically has nothing to do with graphics; in order to run, an appropriate protocol-converter must run on the workstation.

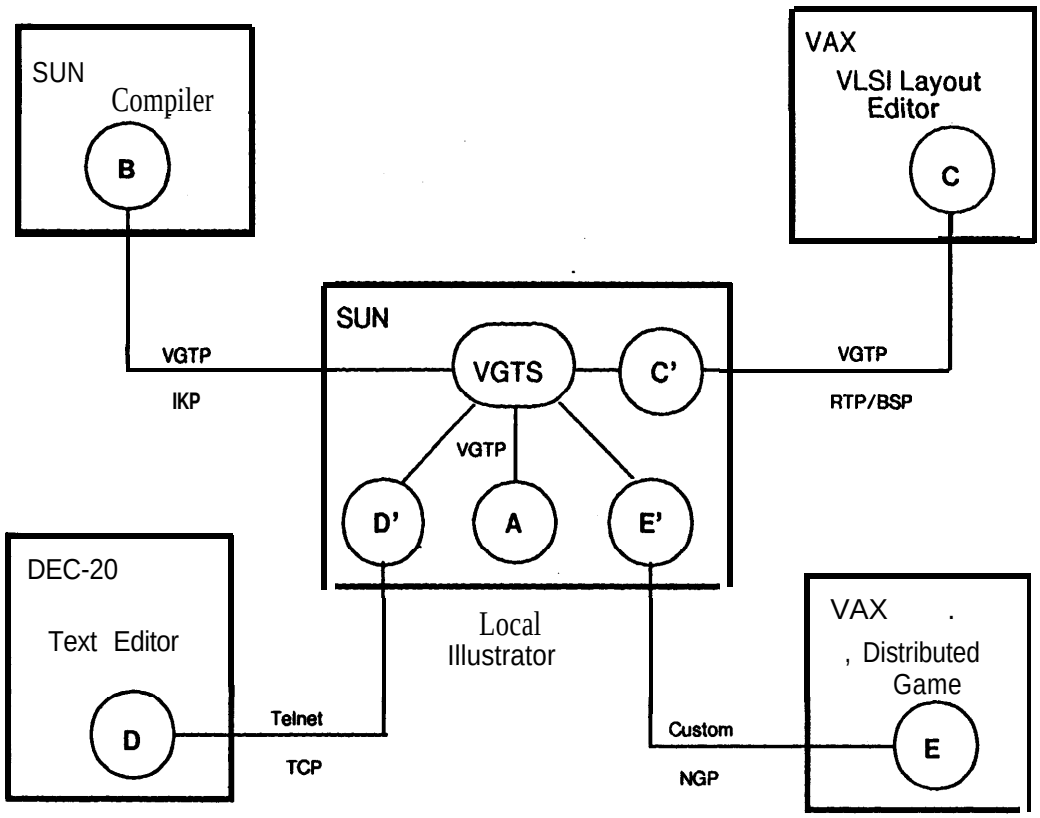

Figure 2-1: Classes of applications in the V-System.

Applications C and *E*, on the other hand, know all about the VGTS and are very interested in graphics. We will refer to the protocol they employ as the *network graphics protocol* (NGP). The NGP may be a simple encapsulation of the VGTP within an existing transport protocol, it may be a problem-oriented protocol [20], or it may itself be a multi-level protocol. Application C, for example, may find a direct encapsulation of the VGTP acceptable. Application *E*, however, may wish to maintain a replicated database (the main database and the cache), or may wish to trade reliability against cost. In these cases, the NGP offers considerably more function than mere encapsulation/decapsulation of the VGTP. In general, the VGTP and NGP correspond roughly to presentation and session layer protocols, respectively, in the ISO reference model [24].

3. Observed Behavior of Distributed Applications

Several distributed applications written specifically for the V-System are graphics-oriented. All graphics operations are performed through the services of the VGTS, which imposes a very definite style of interaction on applications by means of the high-level protocol (the VGTP) employed. Thus, the perspective presented might be regarded as biased towards this one class of applications. However, we claim that the observations presented below are applicable to almost all distributed applications since they can be converted into equivalent non-graphics-oriented statements.

3.1. Nature of the Experiments

Performance measurements have been taken for three benchmark programs, two for graphics and one for text, in a variety of test configurations.

3.1.1. Benchmarks

The first graphics benchmark created a fully-connected 36-agon with a radius of 350 pixels, drawing 630 vectors or 288,364 pixels. Thus the average vector size in this benchmark was 457 pixels. Since the picture was a fully-connected polygon, many different angles of vectors were used. This was intended to test the performance of traditional vector graphics functionality. The action was repeated ten times, and the numbers listed are the mean of ten consecutive trials.

All numbers given as vectors per second in this chapter refer to this same artificial benchmark, so they should be valuable for relative comparisons but not absolute limits. However, since most significant computation was done before the timed parts of this program, and the number of items in the picture is relatively large, the intent was to measure the peak rates of adding items to a symbol and then drawing that symbol. This would measure the rate of initially drawing a new picture.

The second graphics benchmark was intended to test the effects of using structure on a simple picture of the kind used in a VLSI layout editor [7]. This benchmark drew an array of five by six NMOS inverters [13]. Each of these 30 invet-ters consisted of 26 rectangles, for a total of 780 rectangles, all filled with one of four stipple patterns (which would appear as colors in a color implementation) representing the four NMOS layers. First the picture was drawn using a single-level display file and adding all 780 rectangles individually. The second part of this test defined a contact cut symbol, then an inverter symbol, and then added 30 calls to the inverter symbol, with only 23 primitive items in the display file. Although the *regularity factor* of this drawing (the ratio of total items divided by defined items, or 30 in this case) is fairly high, modern VLSI designs typically have regularity factors in the same range, and the trend is to increasing regularity [10, 11]. In fact, many of the designs currently under development would not be possible with smaller regularity factors. Independent of the structure, the resulting image was about 400 pixels on a side.

The text benchmark programs simply wrote characters until stopped by the user. This behavior would occur, for example, when displaying a new page in a text editor. The characters were from a fixed-width font with each character eight pixels wide and 16 pixels high, or 128 total pixels per character. This was the standard font used by most applications except those doing specialized text display.

3.1.2. Test Configurations

The actual structure of the protocols and programs used in the performance measurements are illustrated in Figures 3-l and 3-2. The VGTS is implemented as a collection of server processes within the V-System, access to.which is achieved via-i/O connections. The server host implementations are thought to be representative of what other distributed graphics systems might use. The benchmarks were conducted with the following communication configurations:

- Local — Application running on the same workstation as the one used for display. The application sends V messages directly to the VGTS. Since the application runs in a separate address space from the VGTS, the V kernel's data transfer operations are needed to move information from the application to the VGTS' address space; no shared memory is used. This is illustrated in Figure 3-l a.

- Sun-IKP — Application running under the V-System but on a different machine, connected via Ethernet to another workstation, and using the V-System inter-kernel protocol. As illustrated in Figure 3-1b, the application uses the same message-passing interface, but with the kernels implementing the IKP.

- VAX-IKP — Application running under VAX/UNIX, connected via Ethernet to the workstation, and using V-System IKP. As illustrated in Figure 3-2a, this involves the application writing to a

pipe, which is read by the V-server program, which sends messages over the network to a V kernel. The workstation runs a simple "agent" called **fexecute** which is necessary only because both the VGTS and the V-server are both servers; they both are sent messages to which they reply, instead of initiating the sending of messages by themselves.

PUP Application running under VAX/UNIX, connected via Ethernet to the workstation, and using PUP TELNET. Figure 3-2b illustrates this configuration. The application uses pseudo-tty devices **(ptys)** to communicate with the PUP TELNET server program **(Tel ser).** This program sends packets over the network to the workstation, where a user PUP TELNET program sends the messages to the VGTS. On both ends, the TELNET programs contain (are linked with) the transport- and network-level code.

E-IP Application running under VAX/UNIX, connected via Ethernet to the workstation, and using Internet TELNET. This is Figure 3-2c. The application again uses pseudo-tty devices to communicate with the IP TELNET **server (Tel netd).** The implementation of the transport protocol in this case is in the UNIX kernel on the remote host and in the **internet** server on the workstation. **|The** user TELNET program finally sends the messages to the VGTS.

A-IP Application running under VAX/UNIX, connected via Ethernet **and** ARPANET to the workstation, and using Internet TELNET. This is the same as Figure 3-2c, but the network now includes a gateway and an extension through the ARPANET **backbone.**

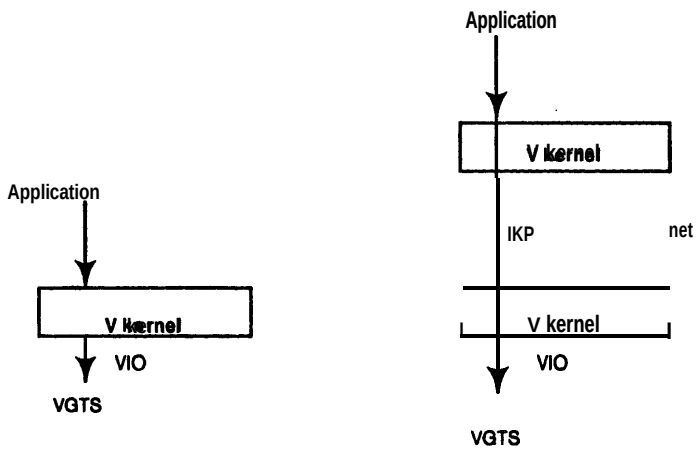

Figure 3- 1: Configurations with hosts running V only.

Tests were conducted using standard **10** Mbit/second Ethernet and 10 MHz 68000's, unless otherwise noted. For configurations involving VAX-11's, 750's, 780's, and a 785 were used, and the tests were conducted at night with correspondingly light loads. Real applications are often run with high timesharing loads, but these are hard to control for the sake of the experiments.

Even more difficult to control were changes to underlying software. Some variation through time inevitably occurred in the VGTS, other workstation software, and host software. For example, introducing new features and fixing errors typically reduce performance, while easing bottlenecks found during experiments improves performance. While each table in this paper compares configurations with similar software, two different tables may compare configurations with dissimilar software. The detailed results presented in [16] specify each configuration.

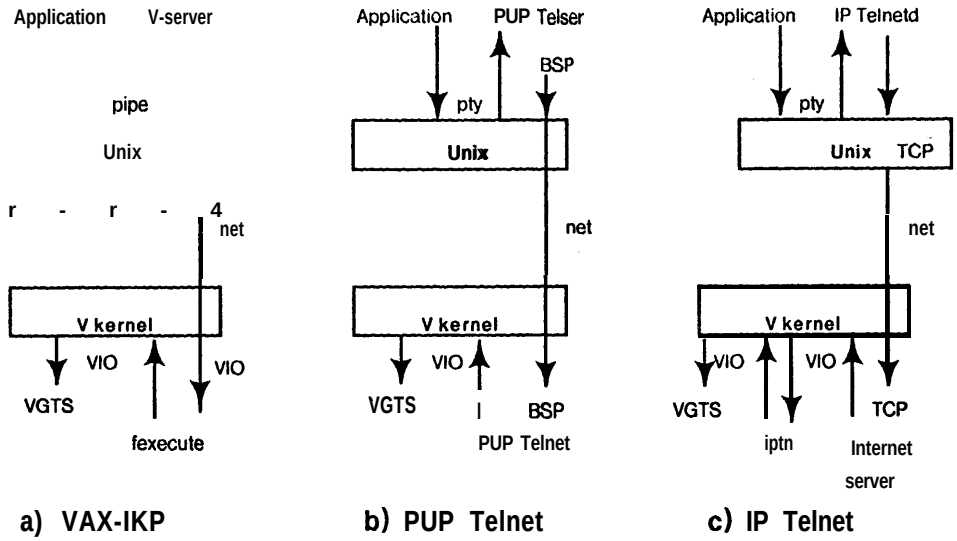

Figure 3-2: Configurations with workstation running V and remote host running UNIX.

3.2. Summary of Performance Results

Given the declarative nature of the VGTP, four measures of interest are:

construction rate — The rate that objects can be added to a symbol, without any display operations.

batch fate — The rate that objects can be added to a symbol, and then displayed.

incremental rate — The rate that objects can be added and displayed as each is added.

display rate — The rate that objects can be displayed out of the display file.

Construction rate is the best measure of the peak network offered load for distributed graphical applications. The batch rate takes into account display overhead, which is fairly independent of the network. Nevertheless, it gives the best measure of overall graphics throughput. On the other hand, the incremental rate gives a better measure of expected response time, when interpreted as the maximum number of display transactions per second. Display rate is another measure of response — to screen rearrangement or simple editing functions. However, it is not affected by distribution and will be little discussed here.

Vector graphics performance is summarized in Table **3-I.** In all of the tables, columns are labeled with the test configurations listed above (local, Sun-IKP, VAX-IKP, PUP, E-IP, and A-IP). Most rows **are** labeled with *(speed, host, rate)* triples, where *speed* is the speed of the Sun workstation processor (8 or **10** MHz), *host* is the type of **VAX (750, 780,** or 785), and *rate* is one of the rates listed above (construction, batch, or incremental). All numbers are in vectors or characters or rectangles per second, so larger numbers indicate better performance. All results have been rounded to two significant digits, and should be taken as order of magnitude estimates only, due to the many factors involved. However, as we shall see, even these very rough measurements can be helpful to determine the feasibility of this approach.

Table 3-I presents the performance figures for configurations employing the most common processors, 10 MHz Sun and VAX-750. As shown by the construction rate row, objects can be constructed at 440 vectors/second for applications running locally, and 380 vectors/second for Ethernet-based applications. Overall graphics throughput, as shown by the batch rate row, is 220 vectors/second for local applications, up to 350 vectors/second for Ethernet-based applications, and 120 vectors/second for ARPANET-based applications. Incremental display permits **62** vectors/second

for local applications, up to 87 vectors/second for Ethernet-based applications, and 39 vectors/second for ARPANET-based applications. Actual display rates are on the order of 430 vectors/second, or .2 million pixels/second, or 5 microseconds/pixel, including all display overhead.

	Vectors/second				
Configuration	Local	IKP	PUP	E-IP	A-IP
10,---, display	430				
10,750, construction	440	380	200	220	130
10,750, batch	220	350	200	220	120
10,750, incremental	62	81	58	87	39

Table 3-1: Summary of vector graphics performance.

The text results are summarized in Table 3-2. Throughput is 7700 characters/second for local applications, up to 4300 characters/second for local net-based applications, and 1900 characters/second for ARPANET-based applications. Additional details appear in Tables 4-l and 4-2.

	Characters/second				
Configuration	Local	IKP	PUP	E-IP	A-IP
10,780, text	7700	4300	1600	4300	1900

Table 3-2: Summary of text performance.

3.3. Evaluation

The remainder of this paper will examine the measurements just presented in some detail. We will attempt to show the effect of varying different parameters, and, ultimately, enumerate the factors in the order of their importance. These parameters include:

- speed of the workstation
- speed of the remote host, if any
- speed of the network
- choice and implementation of network transport protocol
- level at which information is communicated, including characteristics of the high-level graphics protocols

We will demonstrate that performance is most sensitive to processor speed and least sensitive to network speed. Sensitivity is measured by the degree of performance improvement relative to the degree of change in the indicated parameter. Thus, a 50% performance improvement due to a 209% increase in processor speed is much more significant than a 300% improvement in performance due to a 6600% increase in network speed.

It is quite easy to rate the sensitivity to hardware factors. Software factors are another matter; it is easy to measure the absolute performance improvement resulting from a change in software, but quite difficult to measure the degree of that change. Also note that there are limits beyond which changing one factor will not affect performance; for example, a CPU-bound application running on a remote host will be little affected by an increase in workstation speed. Nevertheless, certain conclusions can be drawn based on available information.

There are, of course, other factors that affect performance, including the choice of algorithm and the implementation of "inner loops". For example, recoding the inner loop of the lowest-level line-drawing algorithm improved VGTS display rates by 200% [16]. While the return on investment can be quite impressive, such improvements typically are good for only a onetime increment in performance. This paper focuses on factors that have the potential of influencing performance on a much broader scale.

4. Effects of Processor Speed

As should be expected, the speed of an application is directly related to the speed of the processor on which it runs. Use of 10 MHz Sun workstations instead of 8 MHz workstations yielded up to 22% improvement (see Table 4-1).[2] Use of a VAX-11/780 instead of a VAX-11/750 yields up to 50% improvement (see Table 4-2).

		Vectors/second			
Confiouration	Local	IKP	PUP	E-IP	A-IP
10,780, batch	210	190	130	110	92
8,780, batch	180	150	110	QQ	88
		Characters/second			
10,780, text	7700	4300	1600	4300	1900
8,780, text	6700	3200	1400	3600	1800

Table 4- 1: Effect of workstation speed.

	Vectors/second		
Configuration	IKP	PUP	E-IP
10, 780, construction	510	210	170
10, 750, construction	340	130	110
	Characters/second		
10, 780, text	4300	1600	4300
10, 750, text	4100	1400	2300

Table 4-2: Effect of remote host speed..

Two of the more surprising results relate to the benefits of distributed computing. First, applications can be expected to run *faster* when distributed between a VAX-780 and a Sun workstation than when run locally (see Table 4-3). Although construction rates are lower in the distributed case, the concurrency from the use of two processors results in higher rates for both batch and incremental display. Second, some applications execute faster using a VAX-785 on the ARPANET than using a VAX-750 on the local net (see Table 4-4). Since the ARPANET is substantially slower than the Ethernet and network communication in general is slower than local communication, the conclusion is that CPU speed is the dominant factor in this instance.

	Vectors/second	
Configuration	Local	E-IP
10,780, batch	220	380
10, 780, incremental	62	92

Table 4-3: Sun workstation vs. Ethernet-based VAX-11/780.

Regardless of where the application executes, the workstation is always required to do some work, namely, to maintain and display the graphical objects. Therefore, performance is more sensitive to workstation speed than to remote processor speed. For example, whereas a 25% increase in workstation speed results in almost linear speed-up, a 100% increase in VAX speed results in at most 50% speed-up as seen in Tables 4-1 and 4-2. Note that Tables 4-1 and 4-2 were constructed with early versions of the protocols; later changes to the protocols increased the sensitivity of IP to server host speed, but decreased the sensitivity of IKP and PUP.

[2] **The principal reason that the increase from 8Mhz to 10Mhz 68000 processors did not produce a 25% increase in the performance was that the 10MHz design required polling of the keyboard and mouse.**

Confiauration	Vectors/second E-IP	A-IP
10, 785, construction		160
10,750, construction	130	
10, 785, batch		140
10, 750, batch	125	

Table 4-4: ARPANET-based VAX-11/785 vs. Ethernet-based VAX-11/750.

One might conclude from these measurements that there is little reason to distribute applications, since the proven performance improvements might not outweigh additional programming demands. However, the processors tested differ by at most a factor of 4 in performance. If, on the other hand, an Cray-XMP were available, we would expect a marked increase in performance — roughly linear with processor speed. Moreover, our benchmarks make no significant computational or database demands that would take advantage of faster hosts. Finally, some applications simply cannot run on the workstation, due to memory or language requirements, for example.

5. Issues in High-level Protocol Design

The nature of the applications and of the information they communicate among their distributed parts make the network behave differently from what might commonly be expected. In particular, the use of appropriate "high-level" protocols can reduce the degradation that is experienced between different bandwidth networks. This can influence the choice of remote hosts since the performance penalty of accessing a high-performance host over a long-haul internetwork instead of a less powerful host located on a local network may be outweighed by the difference in host capabilities.

For example, the results for the structured graphics benchmark are given in Table 5-l. The first line shows the performance of a rather simple "stop-and-wait" protocol where every operation requires a return value and no operation is initiated until the response to the previous operation is received. Notice that performance is rather poor, especially over long-delay networks like the ARPANET. Each successive line indicates the performance improvement achieved by adding more "intelligence" to the protocol.

Configuration	**Rectangles/second** Local	E-IP	**A-IP**
10,750, incremental	41	**5**	**2**
10, 750, pipelined incremental	61	66	36
10,750, batch unstructured	310	180	81
10,750, batch structured	1070	**670**	**370**

Table 5- 1: Effects of pipelinina. batching, and structure.

5.1. Pipelining

By removing unnecessary return values, even incremental operations can be pipelined. This results in both fewer messages and more concurrency than the "stop-and-wait" protocol, with an accompanying performance improvement of 50% for local operations and 1000% to 1500% for remote operations (compare the first and second lines of Table 5-l). In fact, remote incremental operations are almost always faster than local incremental operations due to this concurrency.

5.2. Batching

Moving from the second to the third line of Table 5-l shows the effect of another simple change to the graphics protocol. Specifically, for "batch', display, operations are (indeed) batched so that values are returned only after an entire sequence of operations (such as all changes- to a given symbol) have been performed. This change reduces network delays substantially, yielding performance improvements of up to factors of 500% over and above the improvements achieved by pipelining. Other measurements have shown improvements of up to 3000% using only batching. Similar effects can be seen in all tables displaying both incremental and batch rates, where the batch rate indicates the effect of batching and the incremental rates indicates unbatched operation.

5.3. Structure

The virtual graphics terminal protocol allows objects to be defined in terms of graphical primitives such as vectors or rectangles, or in terms of other objects. Once the objects are defined, they can be made to appear on or disappear from the screen with short commands of only a few bytes. This can result in considerable performance advantages, a fact that has, in fact, been known for some time [12].

As seen by comparing the third and fourth lines of Table 5-1, using structure instead of an unstructured list of primitive items increases performance again by factors of 3 to 4 for both local and remote operations. The principal reason for this is that considerably less data needs to be transmitted. The benefits of reducing the amount of data transmitted are even more obvious by comparing the VGTP to protocols that transmit bitmaps — such as Remote BitBlt [8].

The amount of data transmitted is directly proportional to the amount of memory used to store the data at the workstation, so Table 5-2 shows memory usage. In the vector benchmark, for example, the structured display file (SDF) represented the fully-connected polygon with 20 bytes per item, or 12,600 bytes. This compares to the 800 by 800 bitmap area, which would take 80,000 bytes. In practice, most objects are even less dense than the fully-connected polygon, so the advantage would be even greater. In particular, the SDF approach has the advantage as long as there are more that 20 bytes of bitmap space for each item in the SDF. The rectangle benchmark shows that even without using structure, a factor of about two in memory savings is possible. Using structure, the 900 bytes used by the SDF is a factor of 37 less than the space for the bitmap.

	Bytes of memory used	
Benchmark	SDF	Bitmap
vector	12,600	80,000
rectangle, unstructured	15,600	34,000
rectangle, structured	900	34,000

Table 5-2: Effect of structure on memory usage.

5.4. Summary

Ultimately, the time to define and display the picture for a local application was about 1 millisecond per item. This is roughly the time to perform a local Send - Receive - Reply sequence in the V kernel, so any protocol that uses a message transaction for each item will be slower. Moreover, it is faster to run this benchmark over the ARPANET and use structure than it is to run the same program locally and use incremental or unstructured display. The latter is comparable to traditional graphics systems. It is also faster to run the program across the Ethernet and use structure than it is to run the program locally, even with batching.

6. Issues in Transport Protocol Design and Implementation

Regardless of the level of communication, it is important to make the underlying transport protocol as efficient as possible. There are two fundamental, and orthogonal, approaches:

1. Take advantage of the particular network or applications environment by using specialized protocols.
2. Implement the chosen protocol efficiently. For example, make judicious use of kernel support and processes in structuring the protocol implementation.

6.1. Tuning the Protocol

The first approach is exemplified by the V-System's inter-kernel protocol (IKP), which for our applications usually provides significantly better performance than PUP or Internet protocols (see Tables 3-l and 3-2). Remember, however, that these measurements indicate PUP and Internet performance using their respective TELNET protocols as "transport" protocols. IKP's performance advantage should be somewhat less if we were to use the real, underlying transport protocols — RTP/BSP and TCP, respectively. The virtues of IKP have been discussed at length in several previous papers, including [3].

Although specialization at such a low level can achieve significant performance advantages, it is nevertheless mandatory that a general-purpose, internetwork protocol be available to communicate with hosts and networks that do not support the specialized protocol. In fact, an internetwork protocol may yield higher overall performance than a specialized protocol if it allows access to a higher-performance processor than would be available using the specialized protocol. Moreover, internetwork protocols themselves can be tuned to meet different performance objectives. For example, the PUP protocols tend to emphasis response and are relatively "light-weight", whereas Internet protocols are tuned for throughput and are relatively "heavy-weight',.

6.2. Tuning the Implementation

The second basic approach to efficient network transport, namely, efficient implementation, has been a subject of much discussion in the networking community [4]. Indeed, the implementation of a protocol may have a greater effect on performance than any properties inherent in the protocol itself.

For example, much of the advantage Internet protocols have over PUP for text performance is due to the fact that the Internet transport service is implemented inside the UNIX kernel, whereas PUP is implemented outside the kernel as a user process. This results in significant context-switching overhead. Note again that this overhead is most apparent for network-bound, throughput-intensive applications, such as our text benchmark (see Table 3-2).

On the other hand, IKP is also implemented as a process under VAX/UNIX and tends to perform significantly better (for construction and batch rates) than either PUP or Internet. This advantage accrues from the fact that IKP is tuned for operation on local area networks — both in its protocol design as discussed above and in its implementation. For example, the IKP implementation used 1024-byte packets, while both PUP and IP used packets of 100 or 200 bytes. Similar observations relative to TCP are presented in the next section.

Naturally, network services implemented as processes can be significantly affected by the priority at which they run. Table 6-1 indicates the effect of changing the relative priorities of the application program or the TELNET server program. This test was done using the PUP protocol on a local 10 Mbit/second Ethernet. The first column gives the results for normal operation. For the second column, the operating system gave priority to the TELNET server program. Batch performance actually decreased, since more network packets were sent. For the third column, both the application and the TELNET server were given priority, which increased both the batch and incremental rates. However, as shawn in the last column, the best performance was obtained by giving priority to the application.

Confiauration	Vectors/second Normal	Telser	Telser & Application	Application
10,750, batch	170	160	190	200
10,750, incremental	47	48	58	58

Table 6-1: Effect of process priorities.

Another interesting comparison is between remote execution on a timesharing host and execution on another workstation. Table 6-2 displays this comparison. The construction rate is about the same on the VAX/UNIX system and on the V-System. The incremental rates on the VAX/UNIX implementation are very poor without pipelining, due to the fact that the IKP server process (the V-server) is polling every few seconds for output from a pipe, while the other protocols are interrupt-driven.[3] Note, however, that the total batch rate and the pipelined incremental rate are much higher on the VAX than on another workstation. This is due to the fact that there is less concurrency in the remote workstation case, due to the synchronous nature of V-System message-passing. Much better performance could be obtained by replying to the message before it is processed, instead of after the operations are performed.

Confiauration	Vectors/second SUN IKP	VAX IKP
10,750, construction	380	380
10,750, batch	190	350
10,750, incremental	29	4.6
10,750, pipelined incremental	44	81

Table 6-2: Effect of IKP implementation.

7. Issues in TCP Implementation

Our current VGTP can be used with any transport protocol providing a reliable byte stream. This section presents experiences gained from implementing the ARPA TCP protocol for the V-System's internet server. Most of the throughput numbers presented in this section come from experiments that set up a TCP connection between two Sun workstations, each with its own internet server, and then sent bytes back and forth as fast as possible.

7.1. Packet Buffer Allocation

Protocols like TCP control the flow of network traffic by means of a byte-oriented receive window. However, many implementations manage their memory resources in the form of fixed-length packet buffers. This implies that an estimate must be made of the average number of bytes that will arrive in each packet in order to calculate an appropriate receive window size to present. Multiple buffer sizes can alleviate this problem, but may also incur significant overhead costs on some machine architectures. For example, the network device driver may work most efficiently when copying entire packets to a single location in memory.

Buffer compaction isanother approach to the problem of selecting an appropriate receive window size. This involves copying the contents of several partially filled buffers into a single, subsequently full buffer. However, this approach can break down when employed between hosts of significantly different speed such as a workstation and a mainframe computer. The phenomenon that can occur is that the faster host may inundate the slower host with a large number of packets, each containing

[3] The 4.2BSD-based server could be modified to use the select system call, which would eliminate this delay.

only a small number of bytes. The total number of bytes sent is within the limits of the receive window presented; however the total number of packets that must be processed (and compacted) is larger than the number of buffers available. The result is that packets are lost because the receiving host cannot compact the previous packets fast enough to accept the new ones. This causes unnecessary retransmissions along with their associated timeout delays. Depending on the implementation at the sending end this can bring things to an almost complete halt. This phenomenon has been observed regularly with TELNET connections, where terminal output is typically generated one line at a time.

One possible solution to the problem of matching the receive window to the available buffer resources involves utilization of "hints" from higher level protocols about the nature of network traffic that can be expected. These hints can then be used to allocate buffers of appropriate size and number. For example, file transfer applications will generate network traffic consisting of a steady stream of large packets, whereas TELNET-like applications will tend to generate traffic consisting of irregularly generated large packets interspersed with bursts of many small packets. This approach is already used by several implementations of the Xerox RTP/BSP byte-stream protocol [2].

A second observation about packet buffer management deals with the form of memory management used. Personal workstations usually do not have a great deal of spare memory available, so that preallocation of a large pool of buffers is not reasonable unless local secondary storage and virtual memory management are available. Typically, application-level memory management packages such as the C malloc/free routines provide greater functionality than necessary and may also be tuned for a different pattern of memory usage than is generated by networking code. An improvement can be achieved by utilizing a two-tiered scheme where the general memory management routines are only used to obtain "chunks" of several buffers at a time; which are then kept on a separate free list. Choice of chunk size should be such that each network connection requires on average one chunk of buffers. Variations in traffic are then handled by allocating/deallocating additional chunks as needed. Use of this simple scheme yielded an improvement of about 3% in the performance of the internet server.

7.2. Timeouts

One might expect that if a byte-stream protocol is only (or primarily) going to be used within a local network, that protocol need not implement dynamic timeout values for acknowledgement and retransmission of packets in a byte-stream. Dynamic values are clearly needed for efficient performance in an internet environment since the variable number of gateways and networks that packets must traverse can cause significant variations in transit times. However, in the local case, one would expect variations to be much smaller in magnitude, allowing the use of fixed values for timeout intervals. This considerably simplifies the implementation of timeouts and avoids the overhead of continually monitoring the arrival and departure times of each packet (which typically involves an operating system kernel trap to obtain the current time).

Based on these assumptions an early implementation of the internet server used fixed timeout values, with the following effects:

- Mismatched timeout values caused considerable difficulties with file transfer connections, sometimes slowing throughput down by a factor of two or more. It was discovered that the retransmission timeout interval was set too short for the host being sent to, resulting in the retransmission of all packets.
- Similarly, faulty implementations of the protocols on some hosts caused timeout interval mismatches which noticeably affected the delay characteristics of TELNET connections to these hosts. (Quantitative assessment of the delays have not been possible due to the their intermittent nature.)
- In order to avoid unnecessary retransmissions when dealing with (mostly timesharing, heavily loaded, mainframe) hosts that take a long time to process a network packet, the "standard" retransmission timeout was at one point set to a value of 4 seconds. When

this timeout value was used for timing tests run between two workstations the result was a 10% degradation in performance from using a more appropriate value of 1/4 second.

- At one point too short an acknowledgement timing period was used. This resulted in acknowledgements being sent twice as often as necessary, with a degradation in performance of about 1%.

As one can see, different higher-level protocols and different host combinations impose different values to use as optimal (fixed) timeout intervals. Performance considerations here must include both throughput and delay, since some connections are more concerned with short delay than maximum throughput. For example, TELNET connections desire short acknowledgement timeouts in order to avoid unnecessary delays to the user. In contrast, FTP connections are only concerned with throughput and can afford occasional long delays. Short acknowledgement timeout values for these only increase the network control traffic, thereby decreasing throughput.

Similarly, connections that experience a high error rate need a different set of timeout values from those used for connections which run relatively error free. (Periods of high error rate on a local network are not as uncommon as one might expect — see Section 9). Furthermore, the error rate a connection experiences is variable with time, so that a single set of timeout values for a given connection is also not appropriate.

The conclusion one must draw is that a general byte-stream protocol cannot "cheat" by using fixed timeout values even when it is intended to run primarily in a local network setting.

7.3. General Considerations

The experience gained with implementing TCP indicated that it is a very large protocol which is hard to implement correctly. Furthermore, TCP's performance is very sensitive to errors in its implementation. Differences in throughput and delay of up to an order of magnitude were observed as a result of errors occurring in such places as retransmission, receive window management, etc. Also, all of Clark's comments concerning silly window syndromes were observed to be true, since early TCP implementations on our VAX/UNIX and DEC-20/TOPS-20 hosts ignored his suggestions [5].

Our best TCP throughput to date is about 450 Kbits/sec when using packet sizes. of 1000 bytes. This translates to about 18 ms per 1000-byte packet. TELNET throughput at 4300 characters/sec represents less than 10% of this bandwidth (see Table 3-2). The remaining time is taken up by the application generating the text, together with the terminal emulation and display routines within the VGTS. This demonstrates, once again, the *relative* insignificance of transport issues.

8. Effects of Network Bandwidth

All of the above factors combine to render the actual network bandwidth insignificant. Table 8-I shows that although a 3 Mbit/second Ethernet is about 60 times faster than the 56 Kbit/second links used in the ARPANET, using a backend host on the local network yields less than a 50% performance improvement over using a backend host on the ARPANET. In fact, most of the difference in the total batch rate is due to the delay of the ARPANET and intervening gateway, not any bandwidth restriction. Moreover, there was very little measurable performance difference between using the 3 Mbit/second experimental Ethernet rather than 10 Mbit/second standard Ethernet.

	Vectors/second		
Configuration	E3-IP	E10-IP	A-IP
10, 750, construction	220	230	130
10, 750, batch	210	220	120

Table 8-1: Effect of network bandwidth.

These results can be attributed primarily to the level of communication as discussed in section 5, and the conclusion that processor speed is the usual bottleneck. This is consistent with other measurements of Ethernet performance [21] that show very low utilization of the available bandwidth of the Ethernet, and comparatively long delays on the ARPA Network. Thus, these systems rarely approach the limits described in analytical studies that concentrate on performance under heavy loads [22]. In fact, these protocols can be used on very low-bandwidth communication links.

Each ***AddItem*** call sends 20 bytes of data, so a construction rate of 230 items per second (the Ethernet load given in Table 8-1) corresponds to only 4600 bytes per second, or about 40 Kbits/second — 0.4% of the Ethernet's bandwidth. Due to the small amount of data, graphics could even be possible over standard speed telephone lines. For example, at 1200 bits/second, a peak rate of 7.5 items/second should be possible. To test this, the experiment was run successfully on a workstation over a 1200 bits/second telephone link. Several other rates were tested using point-to-point RS-232 connections at various speeds, with the results given in Table 8-2.

	Items/second				
Configuration	1200	2400	4800	9600	10M
10,750, construction	7.4	14	26	54	166
10,750, batch	6.2	12	23	46	131
10,750, structure	84	142	230	320	380

Table 8-2: Effect of point-to-point communication rates.

For the structure benchmark, even at 1200 bits/second, the measured creation rate was 7.4 items/second, very close to the maximum 7.5 calculated above. This rate is slightly less than linear in relation to the bandwidth, indicating that even at low speeds the CPU can be a factor. Moreover, the total rate when using structure was 84 items/second at 1200 bits/second, which is twice as fast as running the program locally with incremental drawing (the first entry in Table 5-l). Structure and lack of significant delays also makes this rate faster than the batch rate for the ARPANET (the last entry in Table 5-l). Finally, the 9600 bits/second structure rate is only about 15% slower than using Ethernet, even though Ethernet has a raw bandwidth a thousand times greater.

9. Other Factors

9.1. Client Packet Size

A dramatic performance improvement was achieved with a special version of the **internet** server that allowed clients to Read and Write blocks of 4 Kbyte rather than the "normal" 1 Kbyte limit. The server divides these blocks into 1 Kbyte chunks for actual transmission over the network. This effectively eliminated an additional Send-Receive-Rep& message exchange per packet (along with the associated overhead of the I/O library routines, etc.) and also produced a mode in which packets would be sent out onto the network in bursts of 4 at-a-time. The result was about a 15-20% increase in performance.

9.2. Remote Internet Servers

Because the V-System supports network-transparent interprocess communication, the **internet** server can be run as a "remote" server, residing only on one host in the local network. This results in significant savings of memory space on "user client" workstations since the **internet** server requires about 60 Kbytes for its code and static data structures and a minimum of 20 Kbytes available for its process stack spaces and packet buffer pools. However, the performance penalty for this configuration can be significant.

To test TELNET throughput, tests were run with a VAX-780 generating text output to Sun

workstations. One connection was opened between the Sun running the internet server and the VAX, and one connection was opened between a Sun not running the internet server and the VAX. By running both connections simultaneously, external factors such as mainframe load variations could be factored out of the performance results. These result showed that performance degraded by about 15% on average for output rates up to about half the maximum throughput capacity of a connection. At higher output rates, the performance penalty rapidly went up, approaching 50% near the connection's peak capacity. This is exactly what one would expect since the network traffic is essentially doubled when the internet server is running on a different host from both ends of a connection, rather than on the same host as one end of the connection. The smaller penalty at lower output rates indicates that performance is being limited by the mainframe rather than by the internet server.

Unfortunately the numbers just presented represent the performance when only two connections are being maintained through the internet server — a local one and a single remote one. However, the whole purpose of running the internet server in a remote configuration is to allow a single copy to service the entire local environment, implying that multiple connections would normally be maintained. Furthermore, the maximum throughput capacity of connections is inversely related to the number of the connections the server is maintaining at any given time. Thus, one would expect to obtain only a small fraction of the performance obtainable with a private copy of the server devoted to maintaining only a single or a few connections. Fortunately, TELNET connections do not normally operate at full throughput capacity, but tend to exhibit a bursty pattern of communications. This implies that the average throughput that the internet server must maintain 'will be a small fraction of the total load that could be presented by all connections combined. Our experience to date confirms this observation.

Accurate delay measurements comparing the local to the remote server configuration have not been obtained due to the difficulty of factoring out random load variations from the results. However, the delay penalty seems to be similar to the throughput penalty in behavior and magnitude.

Similar results were obtained for FTP connections through the internet server to a VAX-780. The average performance penalty for using a remote internet server was about 20%. Unlike the TELNET measurements, this result indicates that performance was being limited primarily by the VAX, once again showing the importance of processor speed.

9.3. Network Pollution

Perhaps the most annoying "external" factor that affects the performance of network communications at Stanford is that of *network* pollution, caused when one or more hosts transmit large numbers of garbage packets. These packets can cause a network's load to increase to the point where the network becomes useless. The problem is further aggravated if the packets being transmitted are broadcast packets, since all hosts on the network must read them in and process them before they can be discarded. Even worse, broadcast addresses on the 3 Mbit Ethernet are frequently designated by the value 0, which is a very common "random" bit pattern! The result is that not only does the network become congested, but all hosts on the network may be brought to a standstill as they spend most of their CPU cycles processing and discarding bad broadcast packets.

9.4. Network Interface Hardware

A final factor that affected performance was the nature of the network interface hardware. Our early-generation Sun workstations use an Ethernet interface that contains a 4 Kbyte hardware buffer for reading packets from the network. Packets that arrive when the buffer is full are discarded by the interface. This effectively limits the length of pipeline that can be specified for a byte-stream connection when dealing with a host that generates packets at a significantly faster rate than the workstation can process them. Consequently, a 10 MHz workstation talking to an 8 MHz workstation can generate packets fast enough so that specifying a TCP receive window larger than about 5000 bytes results in numerous retransmissions due to lost packets. In general, we have observed that

network bandwidth far exceeds interface throughput.

10. Conclusions

First, we have demonstrated that distributed applications *can* run faster than local ones, using common hardware. Indeed, may "real" applications should reap a greater reward from distribution than our benchmarks, since the benchmarks make no significant computational or database demands that would take advantage of faster remote hosts. Moreover, the reader should remember that some applications simply cannot run on the workstation, due to memory or language requirements, for example. Lastly, there is little additional effort in creating an application to run on a remote host — typically, it is simply recompiled.

Second, we have shown that the primary factors affecting performance of our distributed graphics applications are, in approximate order of importance:

1. Speed of the workstation.
2. Speed of the remote host, if any.
3. Level of communication, as determined by the virtual graphics terminal protocol.
4. Choice and implementation of the network transport protocol.
5. Bandwidth of the networks employed.

By modifying point (3), the same observations hold for text. Note again that these observations relate to the degree of performance improvement *relative* to the degree of change in the indicated parameters.

CPU speed rates at the top of the list simply because desired speed-ups can be achieved almost indefinitely by substituting more powerful workstations and backend hosts. Similarly linear improvement is not possible by altering any of the other "variables". The transport protocol IKP, for example, provides as good performance on the local net as can be achieved.

As workstations become more powerful, one might think that offloading functions from hosts to the workstation means that slower backend hosts can be used. In reality, faster hosts are required to keep up with the increased demand of the workstations. On the other hand, many people think that as networks become faster, communication is cheap. Unfortunately, network interfaces have not kept pace with bandwidth, such that many network operations remain CPU-bound. In both cases, the offloading and increased bandwidth allows more users to share the same resource, but does not increase the performance of the individual users. This is true even in the case of microprocessor-based network controllers, which, while intended to improve performance, have in our experience often proven to be slower than simpler and cheaper controllers that perform fewer functions but use fixed logic at a higher speed. Hence, *faster* hosts are needed, not slower ones.

With respect to network bandwidth, sensitivity is directly related to communication requirements. Communications requirements are inversely related to the frequency of communication and the amount of information transmitted, both of which are reduced by the techniques discussed above. Therefore, the remarkable insensitivity of our applications to network bandwidth implies that they are quite sensitive to the "level" of communication.

In our system architecture , the high level of communication is due to the virtual graphics terminal protocol design. In particular, the ability to batch many operations into a single update using a small. number of bytes and the ability to pipeline many of the operations provided large increases in performance. Overall, the use of high-level graphics protocols reduces the degradation that is experienced between different bandwidth networks. This can influence the choice of network transport protocols since the performance penalty of accessing a high-performance host over a long-haul internetwork instead of a less powerful host located on a local network may be outweighed

by the difference in host capabilities. Indeed, the measured performance improvements between transport protocols are considerably less dramatic than those achieved by varying the level of communication.

It is hard to make direct comparisons about network protocols independent of their implementations. For example, a protocol inside the kernel of an operating system is usually more responsive than if it is implemented on top of the kernel. The increased responsiveness comes with the cost of increasing the size of the (usually always resident) kernel and the related difficulties of debugging at lower levels. In our particular case, despite the fact that the PUP protocols are simpler than the ARPA Internet protocols, ARPA Internet-based TELNET connections can sometimes run about twice as fast as Pup-based ones. This is attributed primarily to the fact that PUP is implemented as an application outside the Unix kernel whereas the ARPA Internet protocols are implemented inside the kernel.

With respect to general-purpose internetwork protocols, TCP in particular, we observe that they are complicated and sensitive to subtle implementation details. In order to support differing (and possibly error-prone) implementations on various different hosts one must implement a full "internet-oriented" design. Attempting to simplify various design issues for use only on a local network doesn't work if other hosts use different designs.

References

1. A. Bechtolsheim, F. Baskett, and V. Pratt. The SUN workstation architecture. 229, **Computer Systems** Laboratory, Departments of Computer Science and Electrical Engineering, Stanford University, March, 1982.

2. D.R. Boggs, J.F. Shoch, E.A. Taft, and R.M. Metcalfe. "Pup: An internetwork architecture". *IEEE Transactions* on *Communications* *COM-28*, 4 (April 1980), 612-624.

3. D.R. Cheriton. "The V Kernel: A software base for distributed systems". *IEEE Software* I, 2 (April 1984)' 19-42. .

4. D.D. Clark. Modularity and efficiency in protocol implementation. RFC 817, Network Information Center, SRI International, July, 1982.

5. D.D. Clark. Window and acknowledgement strategy in TCP. RFC 813, Network Information Center, SRI International, July, 1982.

6. J.H. Clark. The Geometry Engine: A VLSI geometry system for graphics. Proc. SIGGRAPH '82, ACM, July, 1982, pp. 127-133. Proceedings published as Computer Graphics 16(3)..

7. T. Davis and J. Clark. YALE User's Guide: A SILT-based VLSI layout editor. 233, Computer Systems Laboratory, Departments of Computer Science and Electrical Engineering, Stanford University, October, 1982.

8. D.P. Deutsch. Design of a message format standard. Proc. International Symposium on Computer Message Systems, IFIP, 1981.

9. K.A. Lantz and W.I. Nowicki. "Structured graphics for distributed systems". *ACM Transactions on Graphics* *3*, 1 (January 1984), 23-51.

10. W.W. Lattin. VLSI design methodology: The problems of the 80's for microprocessor design. First Caltech Conference on VLSI, California Institute of Technology, Pasadena, California, January, 1979.

11. W.W. Lattin, J.A. Bayliss, D.L. Budde, JR. Rattner, and W.S. Richardson. "A methodology for VLSI chip design". ***Lambda (VLSI Design) 2, 2*** (Second Quarter 1981), 34-44.

12. W.D. Little and R. Williams. Enhanced graphics performance with user controlled segment files. Proc. SIGGRAPH '76, ACM, July, 1976, pp. 179-182. Proceedings published as ***Computer Graphics*** 10(2), Summer 1976.

13. Mead and Conway. An *Introduction* to ***VLSI Design.*** Addison-Wesley, 1980.

14. R.M. Metcalfe and D.R. Boggs. "Ethernet: Distributed packet switching for local computer networks". ***Comm. ACM 19, 7*** (July 1976), 395404. Also CSL-75-7, Xerox Palo Alto Research Center, reprinted in CSL-80-2..

15. W.M. Newman and R.F. Sproull. ***Principles of Interactive Computer Graphics.*** McGraw-Hill, second edition 1979.

16. W.I. Nowicki. ***Partitioning of Function in a Distributed Graphics System.*** Ph.D. Th., Stanford University, 1985.

17. J.B. Postel. "Internetwork protocol approaches". ***IEEE Transactions on Communications*** COM-24, 4 (April 1980), 604-611.

18. J.B. Postel. Transmission Control Protocol. RFC 793, Network Information Center, SRI International, September, 1981.

19. D.M. Ritchie and K. Thompson. "The UNIX timesharing system". ***The Bell System Technical Journal 57, 6*** (July/August 1978), 19051929.

20. J.H. Saltzer, D.P. Reed, and D.D. Clark. "End-to-end arguments in system design". ACM ***Transactions on Computer Systems 2, 4*** (November 1984), 277-288. Earlier version appeared in ***Proc. 2nd International Conference on Distributed Computing Systems,*** INRIA/LRI, April 1981, pages 509-512..

21. J.F. Shoch and J.A. Hupp. "Measured performance of an Ethernet local network". ***Comm. ACM 23, 12*** (December 1980), 71 l-721.

22. F.A. Tobagi and V.B. Hunt. Performance analysis of carrier sense multiple access with collision detection. Proc. Local Area Communications Network Symposium, May, 1979.

23. V-System Development Group. ***V-System Reference Manual.*** Computer Systems Laboratory, Departments of Computer Science and Electrical Engineering, Stanford University, 1986.

24. H. Zimmermann. "The ISO reference model". ***IEEE Transactions on Communications COM-28,*** 4 (April 1980), 425-432.