

**SPIMS: A tool for protocol implementation  
performance measurements**  
by  
**Erik Nordmark and Per Gunningberg**

# SPIMS: A tool for protocol implementation performance measurements

Erik Nordmark  
Per Gunningberg  
Swedish Institute of Computer Science

May 24, 1988  
SICS Research Report No: R88006

## Abstract

In the combination of high-speed communication networks and layered protocol implementations, there is a large difference between the performance a user perceives at the application layer and the performance at the physical layer of the network. Still, there exists few general tools to aid a designer in tracing down the performance bottlenecks in the protocol implementations.

In this paper we present such a tool which provides an environment where it is easy to develop, specify, and execute communication 'benchmarks', that can be used to compare the performance of different protocol stack implementations running on different hardware and operating systems. The tool measures user-oriented performance parameters such as throughput and response time without relying on special hardware or operating system support. The measurements are specified in a protocol independent specification language, thus different protocols can be compared by using the same measurement specification.

The tool has been used to compare the performance of the ISODE OSI (FTAM) stack and the Arpa (FTP/TCP) protocols on Sun, MicroVAX and DS90 machines running UNIX.

## 1. Introduction

Earlier measurements on the performance of computer networks were mainly focused on the utilization of shared links between connected sites, e.g. measurements done on the leased lines in ARPANET. Such long haul links normally have bandwidths of 64kbit/second or less, and the cost for these long distance links is a major part of the total network cost and it increases rapidly with both higher bandwidth and longer distances. Hence, the protocols were designed in order to utilize the bandwidth as efficient as possible and network managers tuned the protocols and network configurations accordingly.

A user, i.e. an application process or a person at a terminal, will never get the full bandwidth of the underlying physical link, since the design of the protocols will reduce the bandwidth (headers, flow control et.c.). Indeed, in long haul networks the link bandwidth and the propagation delay are the limiting factors for the user perceived response time as well as for the *actual* bandwidth, i.e. the proportion of the bandwidth that is available for process-to-process communication. Being the limiting factor means that e.g. an increase in the link bandwidth will also give a proportional increase in the actual bandwidth.

In Local Area Networks the user response time and actual bandwidth are *not* limited by the bandwidth of the underlying link. Instead, the performance bottleneck has shifted from the link capacity in the long haul networks, to the protocol and message processing capacity of the communicating computers. Typical, actual bandwidths from file transfer protocol implementations are less than 1 Mbit/second, regardless if the link is 10 or 3 Mbit/second. Thus, faster LAN:s may increase the global volume of data transferred, by allowing more computers to coexist on the network before saturation is reached, while a single user will not see an increase in the communication performance (under operating conditions which are below the saturation point). Still, most measurements and analytical estimations on LAN have either concentrated on the global utilization of the network or on some property of the medium access protocol e.g. access delay and fairness.

Reported results from performance measurements on implementations of protocol stacks are difficult to compare. The measurements are done on different stacks, at different layers, with different message sizes, and in different environments. (As an example of the difficulty, try to compare the results reported in [Strauss87].)

## A new measurement tool

This raises a need of a new measurement tool, which can be used to *compare* implementations of protocol stacks from the *user* point of view, taking into consideration the shifting of the performance bottlenecks from network bandwidth to the protocol processing capability.

Such a tool must also consider that new protocols and applications are emerging in LAN:s. Compared to the typical long haul applications, (i.e. electronic mail, character oriented terminal traffic, and file transfer), the new protocols and applications, (like terminal graphics, network filesystems, remote procedure calls, distributed operating systems, process control et.c.), have much more stringent real-time requirements. A response time in the order of 100 msec is quite satisfactory for a user at a terminal, and most networks are able to exceed this time under normal conditions, while they seldom can meet the response time requirements of remote procedure calls which must be in the order of a couple of milliseconds to be attractive.

At SICS we have designed and implemented a measurement tool with the above considerations in mind. The objectives are that a user of the tool should be able to:

- compare the performance of *whole* protocol stack implementations,
- assess protocol implementations with respect to location of performance bottlenecks,
- easily create communication benchmarks for a desired mix of applications and traffic characteristics, and
- perform distributed measurements without special hardware or operating system support.

The performance results from the tool are throughput and response time. Another result of interest is the host CPU-load the traffic is causing, i.e. what is left-over for other applications. However, this result depends on the availability of operating system support for it.

## The approach

The approach taken is to provide a specification language for the construction of measurements. A measurement specification is a protocol independent way of specifying "what to measure" and "how to measure it". The message specification may include statistical distributions of message sizes and their intermessage sending time, which are complemented with a specification of the environment of the measurement.

Using the language, measurements can be constructed for different types of applications and traffic scenarios. Composite measurements can be constructed in order to create whole communication benchmarks.

The specification technique ensures that benchmarks running on different implementations are identical and hence that the results are comparable. The language and its constructs ensure that benchmarks can be reused for new protocols, and that new benchmarks for new protocols, for different traffic and load scenarios can be designed.

In order to get a tool accepted by the user community and widespread we felt that it must be portable with a minimum of effort and impose a minimum of requirements on the operating system. The requirements using SPIMS include two identical connected computers, and access to the normal operating system clock. Portability is attained by dividing the system into an independent part, consisting of measurement specification and execution control, and a system dependent part. The dependent part hides the process control and has a generic interface to the actual protocol implementations. Thus, the same measurement specification can be used for many protocols, e.g. at different layers.

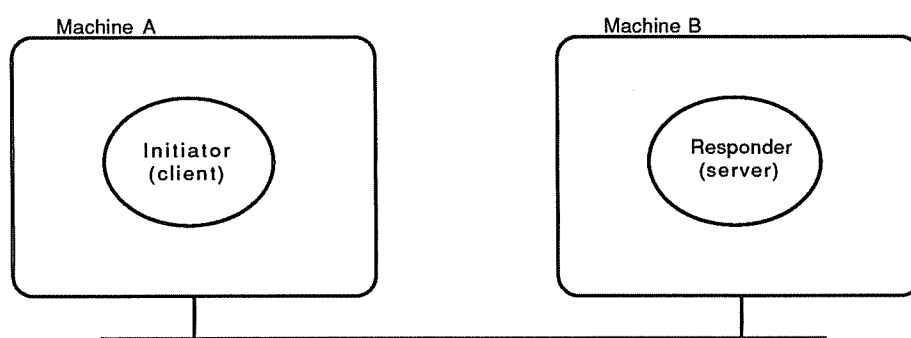


Figure 1 The basic idea behind SPIMS is to measure performance between two identical machines connected by a network, by having a communicating entity on each of the machines. (The *initiator* is the active party in setting up a connection, and the *responder* awaits the connection setup).

## 1.1 Related Work

The standard ANSI X3.102 user performance parameters include quantity parameters such as access time, data transfer rates as well as quality parameters such as the probabilities for incorrect

access, for bit error, lost message, and denial of connection. Many of the parameters are, even though user-oriented, at a low level (Link level). A companion standard, Federal Standard 1043 is describing a method to obtain the X3.102 parameters. In [Seitz83] it is reported on measurements done on ARPANET, using the both standards. The purpose of the measurements were to demonstrate the feasibility of the standards and to propose improvements on the *management* of the network - not the implementation of the protocol - though their conclusion is ..." 'communication software' seems to add the major part of the delay".

Other measurements that also take network managers view of measurement include [Hildago87, Sevcik87].

Some user view measurements, similar to our view, include [Aronoff87, MinnichCotton83, Strayer88, Cabrera87, Cabrera88]. The earlier Minnich-Cotton study on Ethernet controller demonstrates that the performance bottlenecks are located in the higher layer protocols. The Berkeley (Cabrera et al), NBS (Aronoff Mills and Wheatley), and University of Virginia (Strayer and Weaver) studies go into details of the implementations, in order find the constructs causing the bottlenecks. All of them are pointing at implementation issues that are crucial for performance. Berkeley is using short programs measuring throughput and a tool for creating an "execution profile" of the implementation while NBS and UVA are using instrumented implementations with a special common clock. Watson and Mamrak[WatsonMamrak87] are using another approach to find implementation bottlenecks: instruction counting.

## 1.2 Organization

In the following section user-oriented measurements are discussed. Section 3 and 4 describes the specification language, using a generic example, while the measurement environment is discussed in section 5. How the specifications are executed is described in section 6. The results obtained from the tool on SUN, MicroVAX, DS90, with Unix and FTAM, FTP, and TCP can be found in section 4. The paper is concluded with directions for further research and some conclusions. Three appendices are included; on the variance of the results, on the grammar of the specification language, and on the interface to the FTAM protocol.

## 2. User-oriented measurements

The user is not at all interested in the performance of the communication per se! Instead he or she wants his/her batch-oriented programs to execute as fast as possible, and his/her interactive programs should have a fast enough response on user actions. This is also the case when the

program is a file transfer program, or an interactive editor accessing files over the network, or a even a distributed program.

So the user indirectly places requirements upon communication performance, and he/she is interested in the performance under normal operating conditions i.e. including the influence of the operating system, other users et.c.

For applications transferring large amounts of data the execution time depends heavily on the throughput of the communication, and for applications using a lot of remote procedure calls the response time will be critical for their performance. Establishment and termination of connections are other communication services that can be important for the performance of the application.

Clearly the network manager view (how to efficiently utilize the network bandwidth) is the appropriate one when the network bandwidth is scarce, as for slower networks. For these networks there is a strong correlation between the user-oriented parameter "how long time does it take to transfer a 10 kbyte file" and the network-oriented "how many bits have to be sent over a network to transfer bulk data" (which translates into network utilization). But for high-bandwidth networks this correlation is much weaker, and instead the limiting factors are found in the protocol implementations and their interaction with the operating system and the hardware.

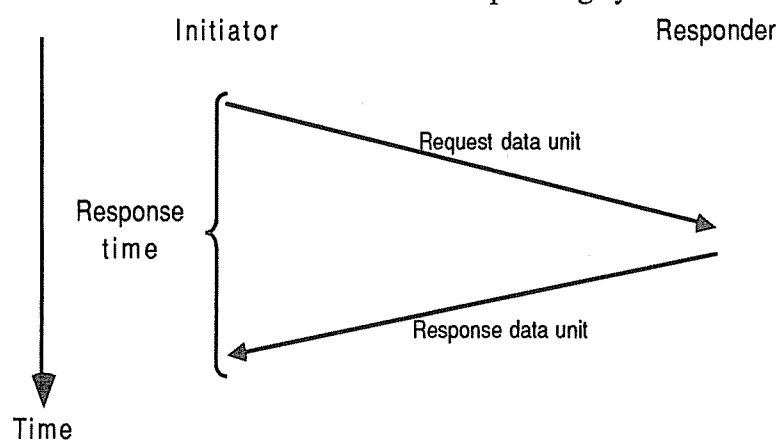


Figure 2 How response time is measured. With *response time* we mean the time it takes from a request data unit is sent from the initiator to the responder until the corresponding response data unit is received at the initiator. (Other terms used elsewhere for this are two-way delay and request-response time.)

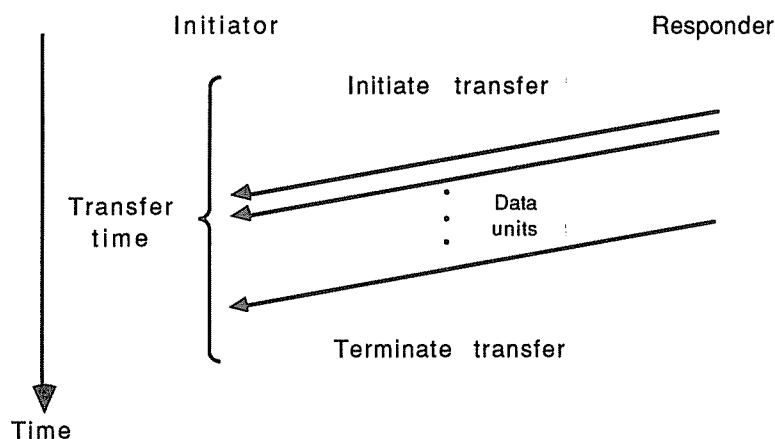


Figure 3 How throughput is measured. We use the term *throughput* for the purpose of our measurements as being the number of bytes transferred between two communicating entities divided by the transfer time, while requiring that there are no data units in transit between the two entities when the "clock" is started and stopped. This requirement has to be handled by the transfer initialization and termination.

### 3. SPIMS measurement specifications

The specification language used in SPIMS is presented by using an example which also have been used to measure the throughput of the FTAM, FTP and TCP protocols presented in Section 7.3. The concrete syntax of the specification language is given in Appendix B, and the complete description of it is presented in [Nordmark88].

#### An example specification

An informal description of a throughput measurement for bulk data transfers is given as:

- Measure the bulk data throughput between application entities on the two machines **khons** and **udjat**, where the RESPONDER sends data units as fast as it can, and the INITIATOR receives them.
- Let the data units have a size of 1024 bytes and measure the time needed for sending 1000 data units.
- Transfer data from *memory to memory*, i.e. factor out the influence of the file system performance.
- Let the data units contain byte strings i.e. the protocols do not need to do any data type conversions.
- Perform 100 measurements and calculate the mean and the variance of the results.

The first step, when specifying a measurement, is to identify the *application type*. With an *application type* we mean the large-scale behavior of the INITIATOR and RESPONDER, and in this case the *application type* is "bulk data transfer going from RESPONDER to INITIATOR". We will use the name **bulk\_get** for this *application type*, and we use this name when we write the specification of the informal description above:<sup>1</sup>

**100 from khons to udjat bulk\_get bytes 1024\*1000 < memory > memory;**

In order to make specifications executable, there has to be an implementation of the application types in which the protocol specific details are hidden.

The second step is to specify how the *application type* can be performed independently of a specific protocol i.e. defining the **bulk\_get benchmark procedures**. An *application type* corresponds to two benchmark procedures, one for the RESPONDER and one for the INITIATOR. The **bulk\_get** INITIATOR should be specified to do the following in order to transfer the data: establish a connection with the RESPONDER, initiate a bulk transfer and (wait to) receive some number of data units. This should be followed by a termination of the bulk transfer and a termination of the connection. Formally specified the sequence of operations at the initiator and responder are<sup>2</sup>:

Name: bulk_get initiator	Name: bulk_get responder
<b>ConnectRequest</b>	<b>AwaitConnectIndication</b>
<b>StartMeasurement</b>	
<b>StartBulkGet</b>	<b>AwaitStartBulkGet</b>
repeat "number of data units" times	repeat "number of data units" times
<b>AwaitDataIndication</b>	<b>DataRequest</b>
<b>StopBulkGet</b>	<b>AwaitStopBulkGet</b>
<b>StopMeasurement</b>	
<b>DisconnectRequest</b>	<b>AwaitDisconnectIndication</b>

Some of the parameters from the specification language, notably the number of data units and their sizes, appear as formal parameter in the *benchmark procedure*. This is not shown here for simplicity.

---

<sup>1</sup>The data type BYTES, and memory-to-memory transfer are defaults, thus they can be omitted from the specification.

<sup>2</sup>The complete specification of the benchmark procedures contains parameters and error handling in addition to what is shown here.

The constructs **StartMeasurement** and **StopMeasurement** defines the period of measurement, and the other bold-faced names in the example are called *communication operations*. They are in a sense an abstraction of the services provided by specific protocols, and they are defined with the intention to hide specifics of the individual protocols but still display the parts of the protocol services that are interesting from a measurement point of view.

The third and final step is to map the *communication operations* to *protocol specific services* (or corresponding), and do this for every protocol that we want to measure. An example of this step is the mapping from the above *communication operations* to service primitives for the FTAM protocol given in Appendix A. FTAM is chosen in the appendix since other protocols, like TCP and FTP, lack abstract service specifications and therefore depend on particular protocol implementations of their interfaces.

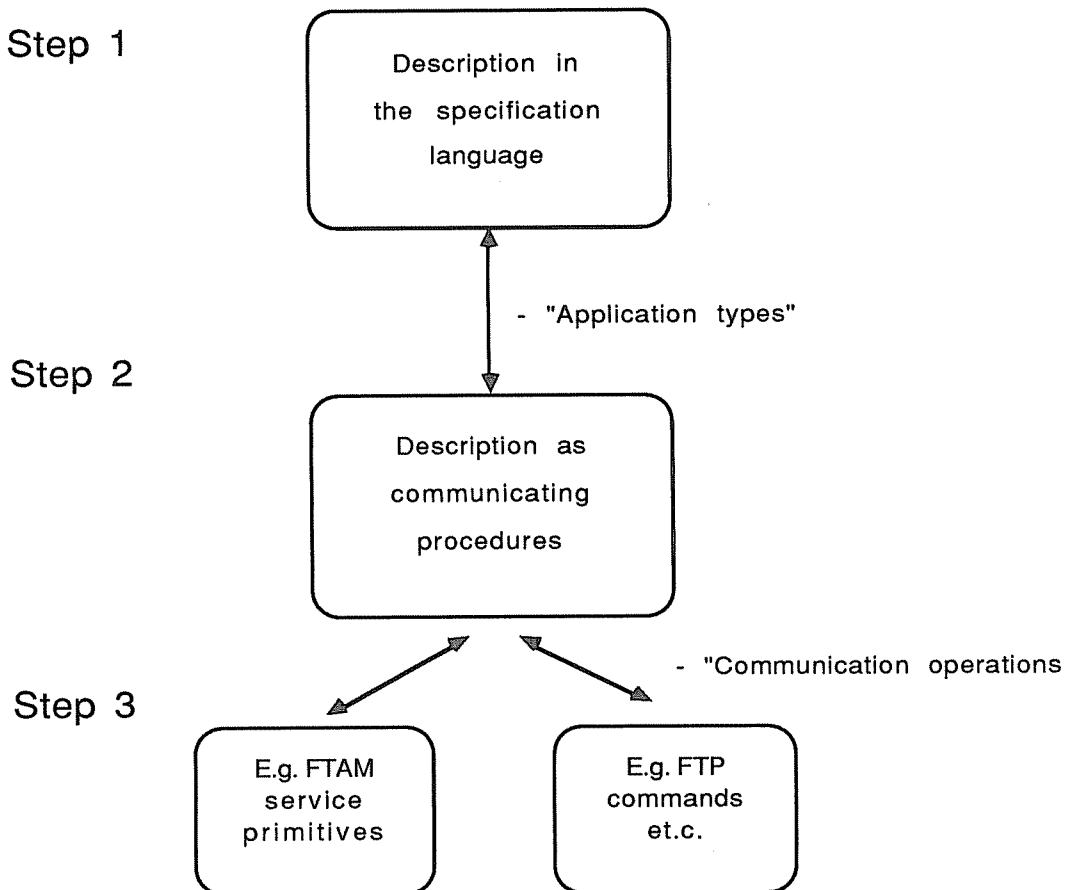


Figure 4 The complete specification. Measurements are specified in a "stepwise refinement" fashion: first in the specification language, then as communicating *benchmark procedures*, and finally as mapping to protocol specific services. The *application type* is the "glue" between the first two, and the *communication operations* between the last two.

The use of *communication operations* instead of the protocol service primitives has the advantage of making most of the specification work reusable. The third step above is done once per protocol. There exists a number of "standard" *application types* (e.g. `bulk_get`, `request_response`, `connect_disconnect`), so that in order to measure on a new protocol it is only needed to implement a basic set of *communication operations* for that protocol. Then all the power of the specification language will be available to measure the performance of the new protocol.

#### 4. Advanced language constructs

The example above demonstrated how measurements of basic parameters such as throughput can be performed. However, the language has more advanced constructs which can be used in order to model the traffic characteristics of other applications such as terminal traffic, distributed operating systems, remote file access with delay due to the disk access et.c. The extensions and examples of their use are given below:

**Distributions on the size of the data units** The data unit size can be a random number for each data unit. The numbers are drawn from a specified statistical distribution, one can e.g. use the bimodal distribution of message size in a distributed operating system to model such traffic[CheritonWilliamson87].

**Time delays - "interarrival time"** Delays can be inserted in the benchmark procedures, where the delay time is a random number drawn from a specified statistical distribution. The delays can be used to simulate e.g. the "interarrival time" of characters from a terminal on the network, or the service time for remote procedure calls.

**Time limited measurements** It is possible to maximize the time of a measurement run instead of specifying the number of data units to be transferred.

**Include file system influence** The data transfers do not need to be memory-to-memory. The benchmarks can e.g. have their received data written to a file. This has been used to compare the throughput of file transfer protocols with the throughput to disk in Section 7.5.

**Parallel measurements**

Several basic measurements, such as the example specification above, can be specified to run in parallel (i.e. multiple INITIATOR-RESPONDER pairs will be executing). This has been used to measure the aggregate throughput in Section 7.5.

**Background load**

For parallel measurements, it is possible to specify that a number of INITIATOR-RESPONDER pairs only exist to create a background load for other pairs. This has been used for the measurements on real-time suitability in Section 7.4.

**Measurements involving more than two machines**

Parallel measurements can be specified to run on more than two machines on the network. It can be used to e.g. perform "stress tests" where one machine receives large amounts of traffic from many other machines.

**5. Environment specification**

It is not sufficient to simply execute the specifications given in the specification language in order to get reproducible and comparable results, since the environment in which the measurements are performed will affect the results.

The factors in the environment that can affect the measurements can be divided into a static and a dynamic part. The former includes the configuration and the parameters of the machines and the network that are being measured. The latter specifies parameters that vary over time such as the load on CPUs and on the network, if pages are being swapped out during the measurement et.c.

The most important static environment factors that effect performance of communication are:

- the network parameters (e.g. bandwidth)
- the network interface (e.g. number of buffers, interrupt frequency [MinnichCotton83])
- the memory (e.g. its size and interaction with the system bus)
- the operating system (e.g. virtual memory support to avoid copying data)
- the processor (e.g. clock cycle, block move instructions)
- the virtual memory
- the file system

The most important dynamic factors are:

- the background load on the network
- the type of traffic on the network (packet sizes, interarrival time)
- the background load on the network interface (e.g. receiving and filtering multicast packets)
- the background load on the system busses (for multiprocessors)
- the background load on the cpu
- the number of idle or almost idle processes on the hosts
- the amount of paging and swapping that the measurement processes are experiencing

Note that the static environment is relatively constant for a given machine, while the dynamic environment could vary on the same machine. This implies that the dynamic environment has to be controlled in order to get reproducible measurements. Thus, an environment specification consists of a listing of all the static parameters, and a listing of requirements on the dynamic ones. The environment specification and the "operational" specification expressed in the specification language defines the measurement sufficiently well so that measurement result will be reproducible and comparable.

#### **A note on benchmarking**

By using the specification language and requirements on the dynamic environment it is possible to specify a set of "communication benchmarks", which can be used to compare the performance of different implementations. Benchmarks consists of a mix of test cases, that should accurately characterize the application and the traffic scenarios.

Benchmarking aiming at estimating the performance of a certain application on different systems must be performed at multiple levels with respect to how "close" the measurements are to the behavior of the application. The levels should range from measuring the times of the primitive operations used by the application, up to executing a stripped down version of the application[Dongarra87] . The appropriate levels for communication benchmarking includes the performance of "primitives" such as bulk data transfer and remote procedure calls, but also the performance when the primitives are used more like they are in the application. This is especially important for communication, since the application's use of the primitives may differ in subtle ways from the way they were measured, and this can affect their performance.

One cause of subtle differences lies in timing considerations, that are not present in non-communicating programs. An example: when using protocols with "timer-based connection management" such as VMTP, the response time can be much higher for the first data unit being

sent after the connection state has timed out, compared to the response time measured at "full blast" i.e. when the connection state will not time out [WatsonMamrak87, Cheriton86].

## 6. Execution of the specifications

In order to get measurement results the specifications has to be executed. In SPIMS this is done by the "executor" which handles two things:

- Controlling the execution of the initiators and responders.
- Collecting measurements.

The control of the execution does not influence the measured performance, and further the interfaces towards the protocols should not add any significant overhead to the measurements.

### 6.1 Measurements

The measurements are taking place during the measurement period specified in the *benchmark procedures* (from `StartMeasurements` to `StopMeasurements`). The elapsed time of this period is the primary metric. It can be measured using a relatively coarse clock granularity, since the specifications should specify an appropriate iteration count.

All measurement are done at the initiator i.e. there is no need to have the clocks synchronized between machines in order to perform the measurements.

Other metrics besides the elapsed time can be gathered depending on operating system support. One, which is supported on most operating system, is the CPU time consumed during the measurement period.

Measurements on identical machines will demonstrate a variance due to normal non-determinism in the execution time, i.e. scheduling, interrupt handling, page set-ups, disk accesses, etc. The variance can be controlled with an appropriate number of runs.

A statistical package calculates the average, standard deviation, minimum and maximum value of all measured metrics. The tool has the possibility to supervise some of the resources of the computer system and signal if something unacceptable happened, like the measurement program being swapped out.

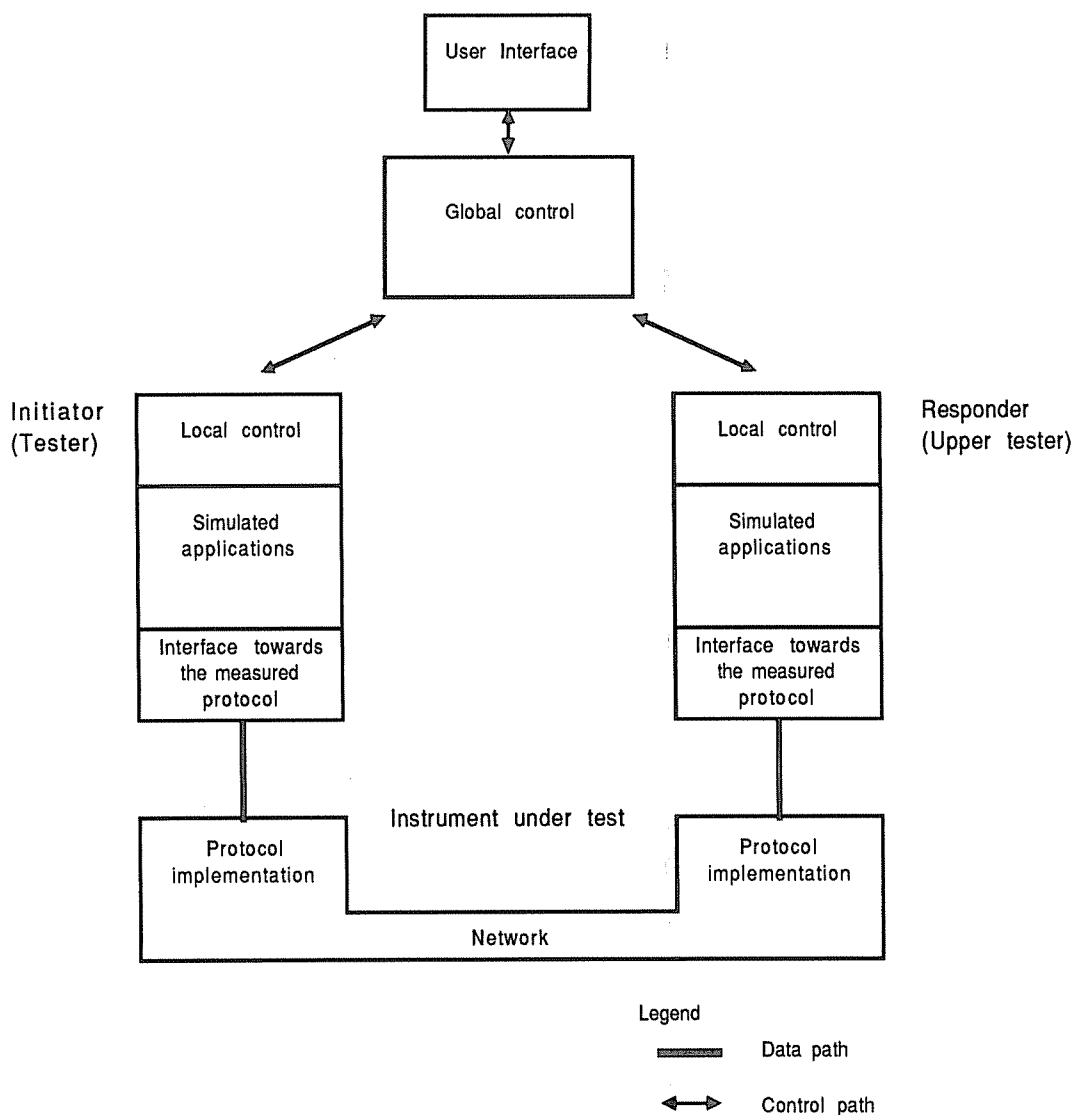


Figure 5 Overview of how measurements are done and how the execution of specifications is controlled. The control "boxes" are state machines communicating using message passing.

## 6.2 Prototype Implementation

SPIMS has been implemented in C under Berkeley Unix™, and it has been ported to a SystemV Unix extended with Berkeley sockets. Figure 6 depicts the modules in the prototype. The specification language is parsed by a YACC grammar. The *benchmark procedures* and the mappings to specific protocols are compiled to avoid unnecessary overhead.

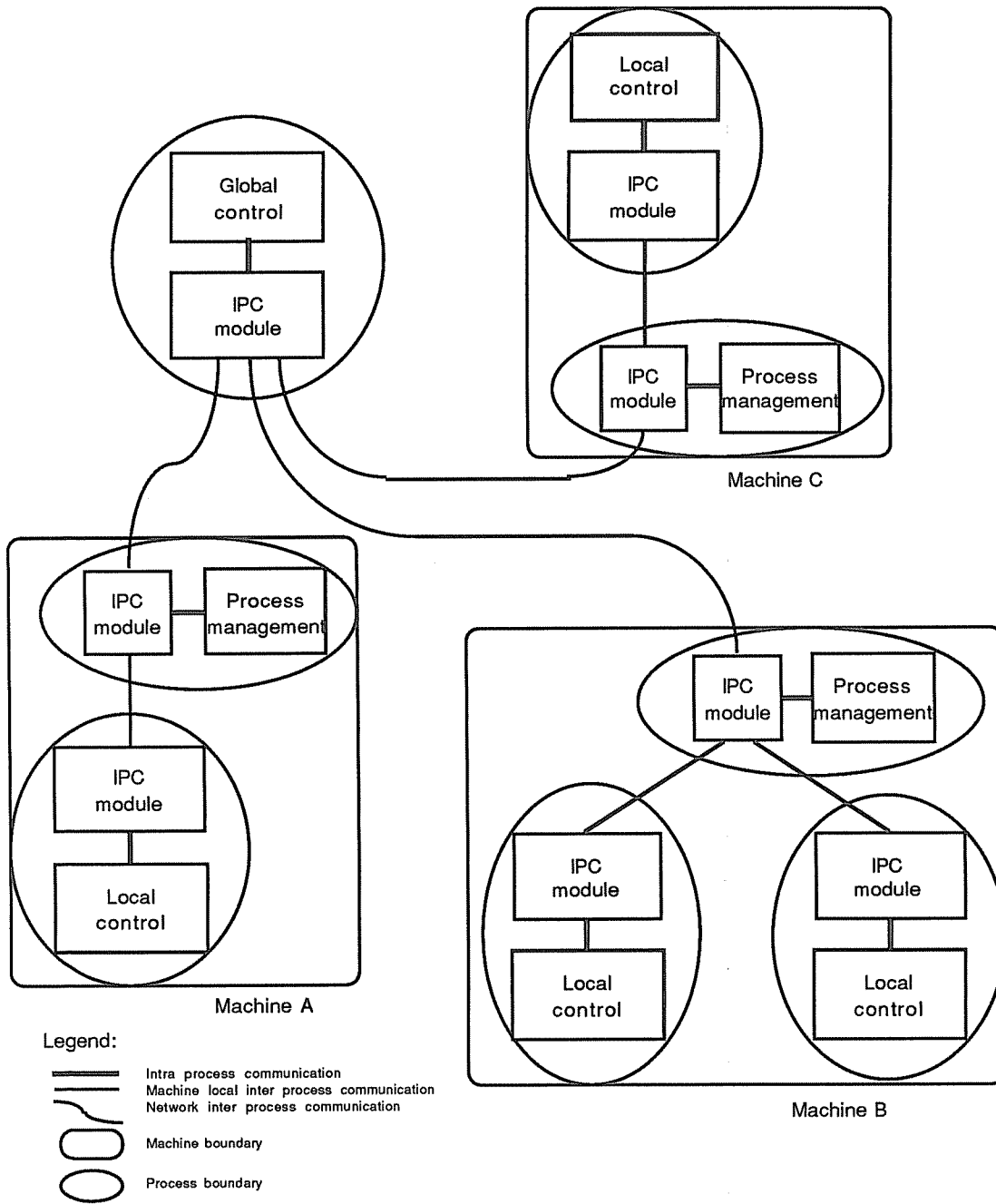


Figure 6 The modules participating in the control of measurement execution in the implementation. The IPC modules provides operating system independent message passing, with network and location transparency, to the control modules. In addition, creation and destruction of processes on the (remote) machines is handled by sending messages to the process management module on that machine.

## 7. Measurement results

In order to illustrate the versatility of SPIMS, we have made some measurements on the ARPA and ISODE protocol stacks on Sun, MicroVAX, and DS90 machines.

The protocol implementations measured are: (1) The DARPA stack (FTP,TCP) supplied with SunOS and Ultrix, and the DNIX FTP and TCP implementations (2) The ISODE OSI stack (FTAM) version 3.0 [ISODE87] running on top of TCP [RoseCass87].

### 7.1 Specifications

The *benchmark procedures* used in the measurements are the following:

**Bulk-get** is the example used in Section 3. It measures the time it takes to transfer a specified number of data units of a specified size, excluding the connect/disconnect time.

**Request-response** measures the time it takes to repeatedly send a (request) message and receive the corresponding (response) message. The request and response messages have the same specified size and the request-response behavior is repeated a specified number of iterations. The connect/disconnect time is not included in the measured time.

**Connect-disconnect** measures the time it takes to repeatedly connect to the responder and then disconnect as soon as the connection has been established.

Figure 7 specifies the benchmarks on the initiator side, i.e. the side from which the benchmark is directed and on which the results are collected, and Figure 8 gives the responder benchmark procedures. The static environment is described in table 1.

Machine	Operating System	Disks	Network
Sun-3/75	SunOS 3.4	NFS	10 Mbps Ethernet
Diab DS90	DNIX		10 Mbps Ethernet
MicroVAX II	Ultrix 2.0		10 Mbps Ethernet

Table 1 The static environment specification for the measured systems

<u>Bulk-get initiator</u>	<u>Request-response initiator</u>	<u>Connect-disconnect initiator</u>
ConnectRequest	ConnectRequest	StartMeasurement
StartMeasurement	StartMeasurement	repeat "number of iterations"
StartBulkGet	repeat "number of iterations"	times
repeat "number of data units"	times	ConnectRequest
times	RPCCall	DisconnectRequest
AwaitDataIndication	StopMeasurement	StopMeasurement
StopBulkGet	DisconnectRequest	
StopMeasurement		
DisconnectRequest		

Figure 7 The initiator benchmark procedures used for the measurements.

<u>Bulk-get responder</u>	<u>Request-response responder</u>	<u>Connect-disconnect responder</u>
AwaitConnectIndication	AwaitConnectIndication	repeat "number of iterations" times
AwaitStartBulkGet	repeat "number of iterations" times	AwaitConnectInd
repeat "number of data units" times	RPCAwaitCallInd	AwaitDisconnectInd
DataRequest	RPCReturn	
AwaitStopBulkGet	AwaitDisconnectInd	
AwaitDisconnectInd		

Figure 8 The responder benchmark procedures used for the measurements.

The dynamic environment can be described as:

- the background load on the network consisted of only periodic updates between the BSD rwho daemons, and periodic routing updates.
- there was no background load on the cpu, except for the periodic sending and receiving of the above updates.
- the "standard" set of system daemon processes where present on the systems. The machines where run in multi-user mode.
- the measurement results where checked so that no paging or swapping had occurred for the processes participating in the measurements.

It has been our experience that the results are not affected if the system daemon processes are removed, since they rarely wake up to perform their system management tasks

## 7.2 Controlling the variance

Based on the measurements presented in Appendix C we made the following compromise between the accuracy of the measurements and their execution time:

- The iteration count was chosen to be 1000 for the comparative measurements on throughput, response time and connect/disconnect time, and these measurements were run 100 times each and the average over those 100 runs was used to compute the results in the comparison.
- For the other measurements we have used an iteration count of 1000, averaged over 10 measurements.
- When the (normalized) standard deviation exceeded 0.05, it is attached to the result.

## 7.3 Comparing the Arpa and ISODE stacks

All measurements are made between two identical machines. The throughput, response time and connect/disconnect time are presented in Table 2 through 4. The specification is given in bold face.

One of the MicroVAX:s that we used in the measurements had a bad Ethernet interface which resulted in the interface being reset when the communication load was high. Therefore, for the MicroVAX, we present both the measured averages and an estimate of what the performance would be with a properly operating Ethernet interface. This estimate is based on the minimum time that SPIMS reported for a measurement run; we observed that the resets only occurred for a portion of the runs, so this estimate should be fairly accurate.

The bulk throughput results presented in Table 1 are just measurements on the (file transfer) protocols without any influence from the file system, and the performance of actual file transfers will of course also depend on the throughput from and to the disk.

Note that the ISO FTAM provides a much larger functionality than the ARPA FTP, and that our measurements are measuring the throughput for byte string data (i.e. no data type conversion) and with a FTAM data unit size of 1 kilobyte.<sup>1</sup>

---

<sup>1</sup>The FTAM initiator and responder programs supplied with ISODE negotiates the data unit size, and on our LAN they use 64 kilobyte data units, which results in less overhead thus higher throughput.

#### 7.4 Simulating real-time communication with background load.

The influence of parallel bulk-get data transfers, running as background load, on the performance of a response time measured by a request-response benchmark, see diagram 1 and 2. All the processes have the same CPU execution priority in one case and the request-response initiator and responder has a higher priority in the other case. The measurement was made on the TCP protocol between two Sun machines.

### 8. Conclusions

We have presented SPIMS, a tool that provides an environment for performance measurements.

Its primary advantages are:

- That it can be used to compare the performance of protocol implementations.
- A specification language for specifying communication measurements.
- The ability to measure user-oriented parameters without relying on special hardware for neither clock synchronization nor collection of measurement data.

The presented benchmark examples demonstrate how versatile and flexible the tool is in specifying benchmarks that can capture the characteristics of a wide variety of applications, protocols, and traffic characteristics. Also demonstrated is the ability to use the same specifications for several protocols.

From the measurement results it is observed that there is a significant performance penalty for the OSI FTAM implementation (of ISODE), compared to the ARPA FTP protocol on a Sun workstation. (However, ISO FTAM provides a much larger functionality than ARPA FTP, and the measurement specification that we used uses a relatively small data unit size.)

Another observation is that it is possible to run the measurement while other light computing and network traffic were running, provided an sufficient number of iterations were used, and that the the results are averaged over a reasonable number of measurement runs.

We plan to use SPIMS to compare several other implementations and protocols, like the high-performance protocols VMTP and XTP, which are claimed to be appropriate for real-time applications. MMS (the Manufacturing Message Standard), especially its ability to handle alarm type of messages with background traffic is another candidate for measurements.

With the tool we will compose an initial set of "standard communication benchmarks" that can be used to get a rough idea of the performance of different protocol implementations.

One direction for further research is to integrate the tool with protocol conformance testers, so that performance specifications can be validated. Some of the features of the tool, such as statistical distributions of the size of data units sizes, and delays with statistically distributed time, have yet to be evaluated with respect to the accuracy of the obtained results.

### Acknowledgements

Kevin Mills at NBS is acknowledged for encouraging us to start this activity on a measurement tool for protocol implementations. Richard Colella, NBS, gave us valuable insight into their simulation and measurement tool. Peter Sjödin has given valuable comments during the design and did a careful proofreading of the paper. A special thanks goes to DIAB letting us use their DS90 computer for our measurements.

### References

- [Aronoff87] R. Aronoff, K. Mills, and M. Wheatly, "Transport Layer Performance Tools and Measurements", *IEEE Network*, Vol. 1, No. 3, July 1987
- [Cabrera87] L-F. Cabrera, "Improving Network Subsystem Performance in a Distributed Environment. A Berkeley Unix™ Case Study", Research report RJ 5719, IBM Almaden Research Center, June 1987
- [Cabrera88] L-F. Cabrera, E. Hunter, M.J. Karels, and D.A. Mosher, "User-Process Communication Performance in Networks of Computers", *IEEE Trans. on Software Engineering*, Vol. 14, No. 1, Jan. 1988
- [Cheriton86] D.R. Cheriton, "VMTP: A Transport Protocol for the next Generation of Communication Systems", *Proc. of SIGCOMM '86*, ACM, Aug. 5-7, 1986
- [CheritonWilliamson87] D.R. Cheriton, and C. L. Williamson, "Network Measurement of the VMTP Request-Response Protocol in the V Distributed System", Computer Science Department, Stanford University
- [Chesson87] G. Chesson, "The Protocol Engine Project", *Unix Review*, Sept. 1987
- [Dongarra87] J. Dongarra, J.L. Martin, J. Wolton, "Computer benchmarking: paths and pitfalls", *IEEE Spectrum*, pp 38-43, July 1987
- [Gunningberg87] P. Gunningberg, "Innovative Communication Processors: A Survey", Research Report SICS R87003, Swedish Institute of Computer Science, March 1987

- [Hildago87] M.B. Hildago, W.S. Kelly, and T.N. Washburn, "Verification/Validation of Performance Simulation using ANSI X3.102 Measurements", Proc. Symp. on the Simulation of Computer Networks, Colorado Springs, Colorado, 1987
- [IlyasMouftah85] M. Ilyas, and H.T. Mouftah, "Performance Evaluation of Computer Communication Networks", *IEEE Communications Magazine*, Vol. 23, No. 4, pp. 18-29, April 1985
- [ISODE87] "ISO Development Environment at NRTC", Northrop Research and Technology Center, Palos Verdes Peninsula, CA
- [MinnichCotton83] N.M. Minnich, and C.J. Cotton, "An Evaluation of two Unibus Ethernet Controllers", 8th Conf. on Local Computer Networks, Minneapolis Minn. , pp. 29-36, October 1983
- [Nordmark87] E. Nordmark, "VMTP Implementation Notes", Stanford University, Computer Science Department, June 1987
- [Nordmark88] E. Nordmark, "SPIMS: SICS Protocol Implementation Measurement System - User Manual", Swedish Institute of Computer Science
- [RoseCass87] M.T. Rose, and D.E. Cass, "OSI Transport Services on Top of the TCP", *Computer Networks and ISDN Systems*, North-Holland ,Vol. 12, 1987, pp. 159-179
- [Sevcik87] P. Sevcik, L. Korn, and J. Niedercord, "Neptune Performance Measurement Concept and Tool", Proc. MILCOM '87
- [Seitz83] N.B. Seitz, D.R. Wortendyke, and K.P. Spies, "User/Oriented Performance Measurements on the ARPANET", *IEEE Communications Magazine*, Vol. 21, No. 5, pp. 28-44, Aug. 1983
- [Spector82] A.Z. Spector, "Performing Remote Operations Efficiently on a Local Computer Network", *Comm. of the ACM*, Vol. 25, No. 4, pp. 246-260, April 1982
- [Strauss87] P. Strauss, "OSI throughput performance: Breakthrough or bottleneck?", *Data Communications*, pp. 53-56, May 1987
- [Strayer88] W.T. Strayer, and A.C. Weaver, "Performance Measurement of Data Transfer Services in MAP", Computer Science Report No. TR-88-04, University of Virginia, Charlottesville, Feb. 1988
- [WatsonMamrak87] R.W. Watson, and S.A. Mamrak, "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices", *ACM Trans. on Computer Systems*, Vol. 5, No. 2, pp. 97-120, May 1987

## Measurement Results

Machine	FTAM	Ftp	Tcp
Sun3	49	239	245
DS90		55	58
MicroVAX <sup>1</sup>	15	50	51
MicroVAX estimate	29	121	133

Specification: 100 bulk\_get 1024\*1000

Table 2 Throughput in kbyte/s. Memory to memory bulk-get using a Data Unit Size of 1 kbyte.

Machine	Tcp
Sun3	7.2
DS90	32
MicroVAX	29
MicroVAX estimate	10

Specification: 100 request\_response 32\*1000

Table 3 Response time in milliseconds. Request-response communication with a 32 byte message size (for both request and response messages).

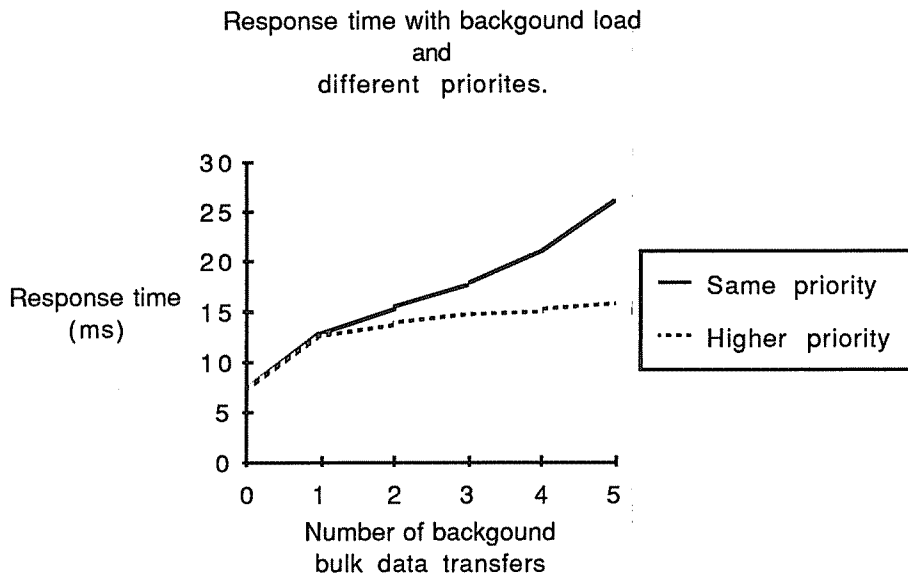
---

<sup>1</sup>One of the MicroVAXes had a "bad" Ethernet interface. Under heavy load the interface hung and was reset. This caused the average to be rather high, while some measurement runs ran without any resets. The *estimated* numbers are based on the results of the measurement runs that did not experience any resets.

Machine	FTAM	Ftp	Tcp
Sun3	853	29	29
DS90		1205	1210
MicroVAX	1220	26	26
MicroVAX estimate <sup>1</sup>			

Specification: 100 connect\_disconnect 0\*1000<sup>2</sup>

Table 4 Time for connection plus disconnection in milliseconds.

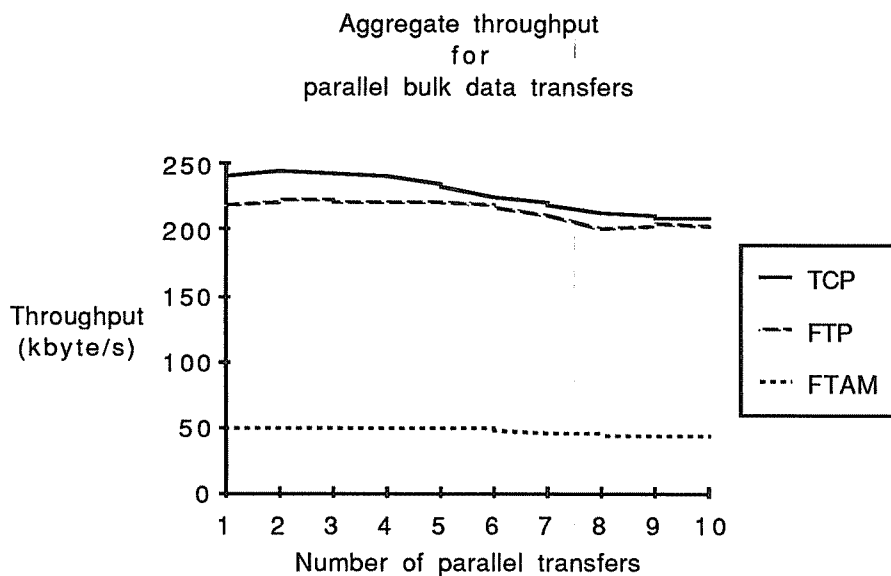


Specification: 10 background: request-response 32 \* 1000 |  
 <N> bulk\_get 1024 \* 3000; N = 0...5  
 and  
 10 priority background: <as above>

<sup>1</sup>This measurement did not load the Ethernet interface enough to cause it to hang.

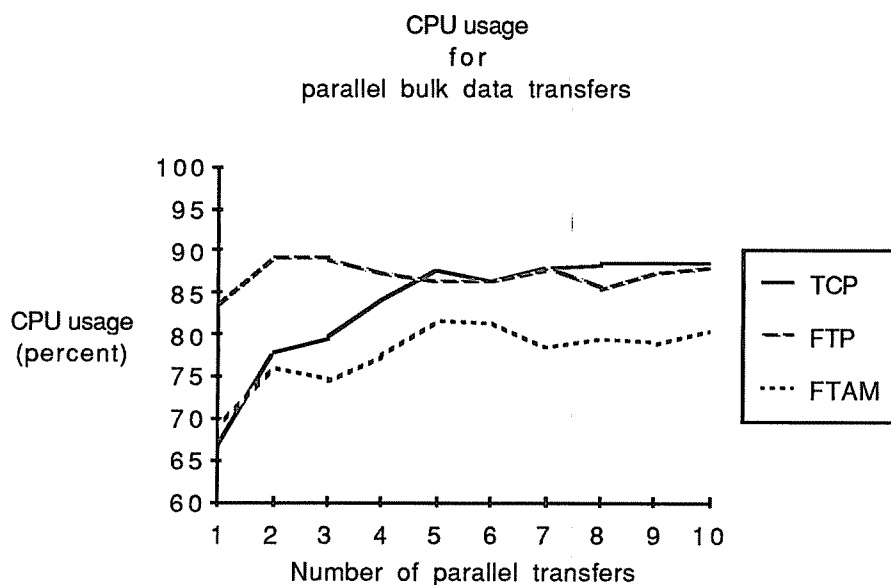
<sup>2</sup>For FTP and TCP on the MicroVAX the specification was run 20 times. This was necessary since the other specification resulted in the machine crash (due to lack of buffer space in the OS kernel).

Diagram 1 "Real-time" request-response communication with background load.



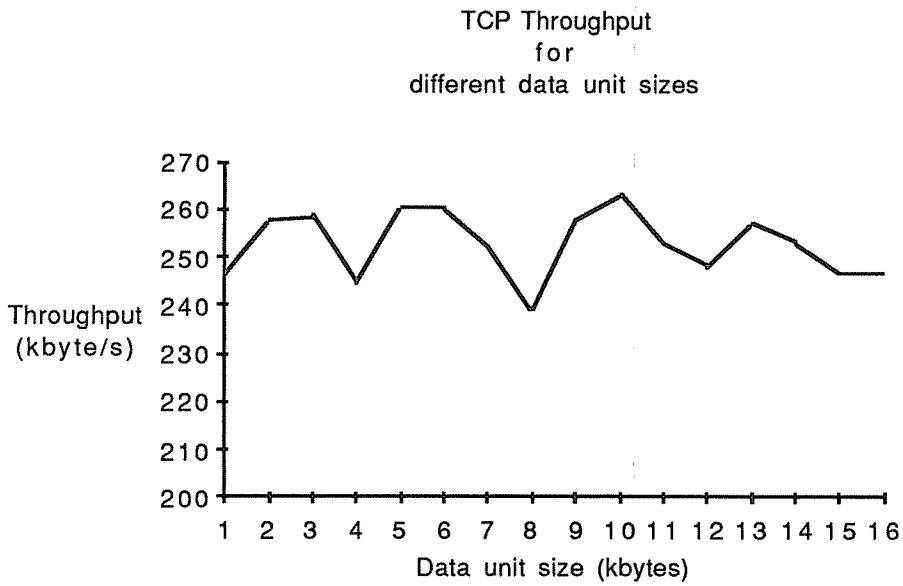
Specification: 10 parallel : <N> bulk\_get 1024\*1000; N = 1...10

Diagram 2 How concurrency influences bulk throughput. Memory to memory bulk-get transfers between two Suns . The aggregate throughput for all the initiator-responder pairs is displayed. (Other parameters as above)



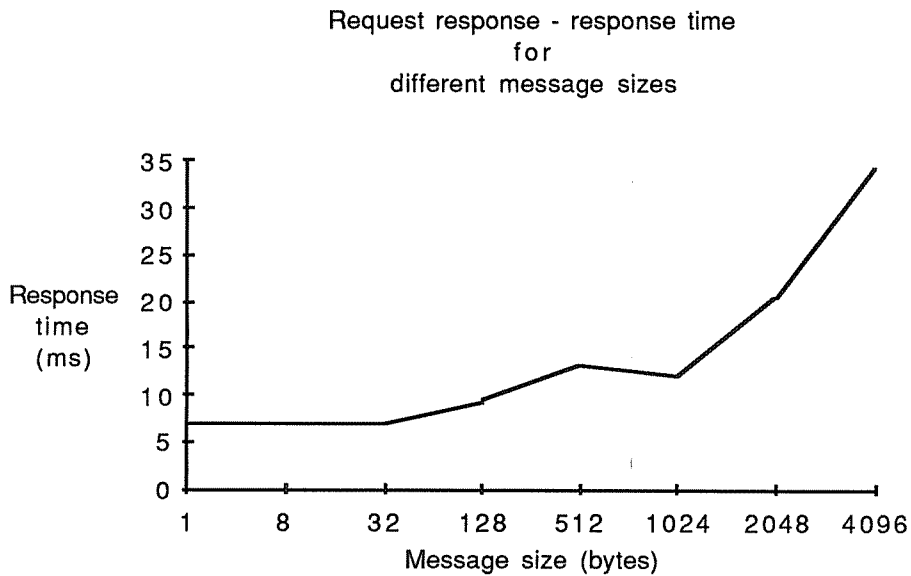
Specification: same as for Diagram 2.

Diagram 3 The CPU usage has been computed by dividing the aggregate CPU time with the elapsed time of the parallel bulk-get benchmarks.



Specification: 10 bulk\_get <N>\*1000; N = 1024, 2048...16348

Diagram 4 Influence of data unit size size on throughput. TCP protocol between Suns.



Specification: 10 request-response <N>\*1000; N = 1, 8, 32...4096

Diagram 5 Influence of message size on response time. Note that both the request and the response message have the same size.

## Appendix A - Initiator interface towards FTAM

Here we describe the mapping from the protocol independent operations used in the example in section 3 to service primitives in the FTAM protocol. The purpose of this appendix is to illustrate the concept so only the names of the service primitives are listed and not their parameters.

We only present the mapping for the *communication operations* used at a **bulk\_get** initiator. There is a corresponding interface for the responder side, as well as for other *application types*.

For those unfamiliar with FTAM we note that REQUEST and RESPONSE primitives are sent whereas INDICATION and CONFIRMATION are events occurring at the receiver of a REQUEST and a RESPONSE, respectively.

Protocol independent Communication operation	FTAM Service Primitive
ConnectRequest	F-INITIALIZE.REQUEST F-INITIALIZE.CONFIRMATION
StartBulkGet	F-BEGIN-GROUP.REQUEST F-SELECT.REQUEST F-OPEN.REQUEST F-END-GROUP.REQUEST F-SELECT.CONFIRMATION F-OPEN.CONFIRMATION F-READ.REQUEST
AwaitDataIndication	F-DATA.INDICATION
StopBulkGet	F-DATA-END.INDICATION F-TRANSFER-END.REQUEST F-TRANSFER-END.CONFIRMATION F-BEGIN-GROUP.REQUEST F-CLOSE.REQUEST F-DESELECT.REQUEST F-END-GROUP.REQUEST F-CLOSE.CONFIRMATION F-DESELECT.CONFIRMATION
DisconnectRequest	F-TERMINATE.REQUEST F-TERMINATE.CONFIRMATION

Note: The FTAM interface relies on synchronous event notification so e.g. `AwaitDataIndication` blocks waiting for an event of the type `F-DATA.INDICATION`

## II

### Appendix B - Specification language grammar

#### Productions:

```
<specification list> ::= <specification>
                       | <specification> <specification list>
<specification>      ::= <num runs> [<timelimit>] <measurement> ","
<timelimit>         ::= "TIMELIMIT" <seconds>
<measurement>       ::= <basic measurement>
                       | <composite measurement>
<composite measurement> ::= <composer name> ":" <part list>
<part list>         ::= <part>
                       | <part> "|" <part list>
<part>              ::= [<num replicas>] <basic measurement>
<basic measurement> ::= <host spec> <application type name> <data spec> <timing list>
<host spec>         ::= ["FROM" <hostname>] ["TO" <hostname>]
<data spec>         ::= [<data type>] <size spec> <source> <destination>
<size spec>         ::= <data unit size> "*" <num data units> |
                       <data unit size> "," <data unit size> "*" <num data units> |
                       <data unit size> [<amount of data>]
<data unit size>    ::= <integer>
                       | <distribution>
<source>            ::= *empty*
                       | "<" <access type>
<destination>      ::= *empty*
                       | ">" <access type>
<access type>       ::= "FILE" <file name>
                       | "MEMORY"
                       | "VIRTUAL MEMORY"
<timing list>        ::= *empty*
                       | <timing> <timing list>
<timing>             ::= <distribution>
<distribution>      ::= <distribution name> "(" <parameter list> ")"
<parameter list>    ::= <parameter>
                       | <parameter> "," <parameter list>
```

#### Pseudo-terminals:

<num runs> the number of times the measurement will be run. The results of each run are given to the statistical package for processing.

<seconds> the maximum number of seconds the measurement will run. (Integer or decimal number) Default: unlimited

<composer name> the name of a constructor of parallel measurement. Examples: **parallel**, **background**, **total**

<num replicas> the number of parallel instantiations of this basic measurement. Default: 1

<application type name> the name of a measurement. Examples: **bulk\_get**, **request\_response** (**rr** for short), **connect\_disconnect** (**conn\_disc** for short)

<hostname> the name of a machine to execute the measurement.

<data type> the type of the data being transferred. Only BYTES (meaning strings of bytes), has been defined so far.

<data unit size> the size of the data unit to be used at the service access point.

<num data units> the number of messages that are sent as part of one measurement.

<amount of data> the number of bytes to transfer. Default: a very large number!

<file name> an operating system file name.

<distribution> a random distribution function such as **constant**, **uniform**, **normal**, **exponential**, **bivalue** et.c.

<parameter> a floating point number. The interpretation of the parameters depends on the distribution function.

#### Legend:

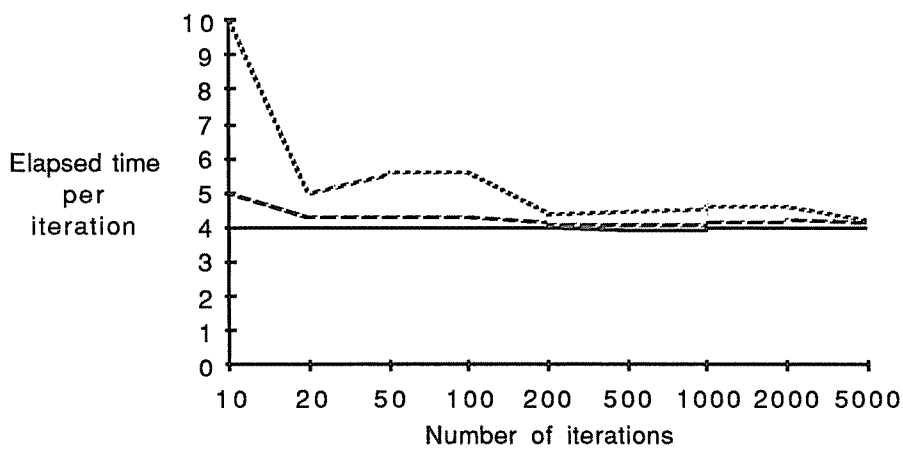
- Non-terminals and pseudo terminals have their names enclosed in <>.
- Terminals are written between quotes and with capital letters.
- \*empty\* denotes the epsilon symbol.

**Appendix C - Controlling the variance in the results**

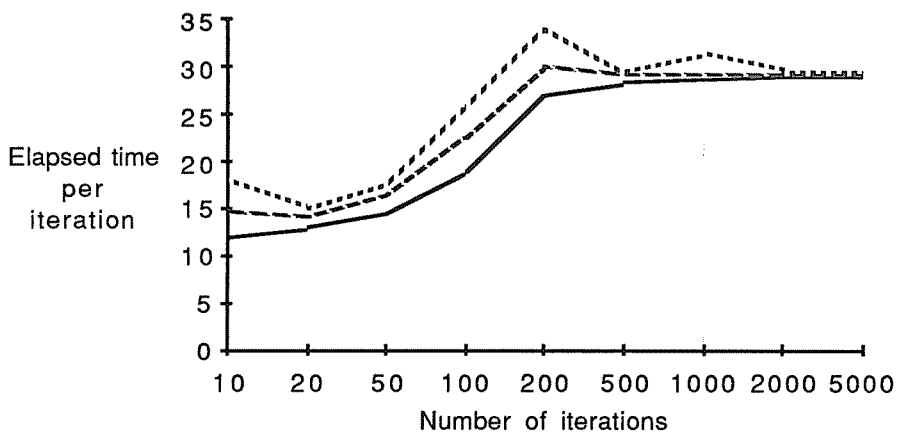
Measurements was made studying the gap between minimum and maximum values and the standard deviation for different iteration counts. The measurements to determine the minimum iteration count were done between two Suns using Tcp. They were run 10 times and SPIMS' statistical package reported the average, min, max, and standard deviation.

This process was repeated for the three application types that we have used, and the iteration count ranged from 10 to 5000.

Bulk get - minimum, average, and maximum  
for  
different number of iterations

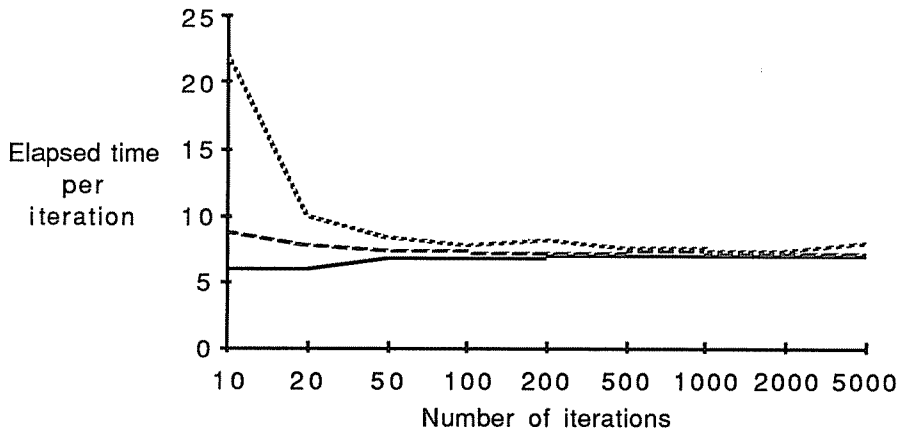


Connect disconnect - minimum, average, and maximum  
for  
different number of iterations



IV

Request response - minimum, average, and maximum  
for  
different number of iterations



To be able to more exactly capture the variance we look at the normalized standard deviation (which is defined as the standard deviation divided by the average value). The diagram below shows that for above then 200 iterations the normalized standard deviation is below 0.05.

Normalized standard deviation  
for  
different number of iterations

