Parallel Communication

Joseph D. Touch

USC / Information Sciences Institute (touch@isi.edu)

Abstract[1]

Advances in communication rates exceed the ability of a single source to fully utilize a gigabit WAN channel with existing deterministic protocols. Parallel communication describes a method for reducing latency by managing indeterminism and increasing channel utilization, given a surplus bandwidth-delay product. It involves a nondeterministic state mechanism, with a modified protocol interface.

1: Introduction

Recent advances in communication rates have (and will) outpace the ability of a single source to effectively utilize channels. This surplus bandwidth, in the form of an excess bit latency (i.e., bandwidth-delay product), provides an opportunity for increased channel utilization. Here we present one paradigm for using such an opportunity, called Parallel Communication.

Parallel Communication extends the channel functions, to permit more responsive user service in domains with high bit latency, through increases in the effective channel utilization. Parallel Communication specifies the ways in which *sets of communication* are managed together, and describes a modification of our existing protocol paradigms that can better utilize links with high bit latencies.

2: Characteristics of Gigabit networks

Gigabit networks are characterized by a high bandwidth-delay product (we prefer bit latency, the latency measured in bits, rather than time), and by the relationship of this product to the messages exchanged. In Figure 1, the equivalences between various speed and scale networks are shown. The bit latency characterizes the protocol operation [9]. Since a gigabit LAN has a bit latency equivalent to a 100 kilobit WAN (ignoring topology issues), TCP/IP (which has worked on WANs up to 1.5 Mbps) should suffice.

All existing protocols incur several trips worth of latency during connection setup. Existing protocols also attempt to pipeline the data transfer during the connection as well. Here we are considering only the mid-stream utilization (ignoring setup) and including conventional pipelining. By these measures, existing protocols, which work in WANs up through 45 Mbps, will suffice in LANs through 100 Gbps (and MANs through 10 Gbps) (see the 45 Mbps WAN rate line in Figure 1).

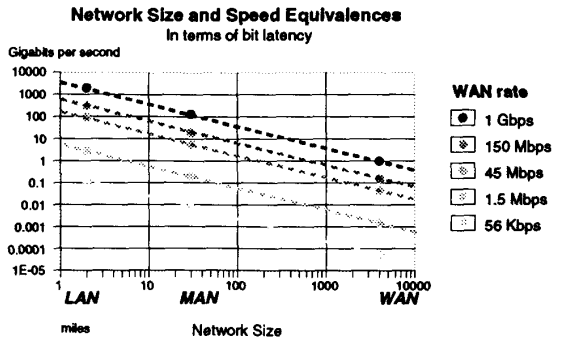

FIGURE 1. Network equivalences. (diagonal dashed lines specify bit latency equivalence)

Bit latency has exhibited three phases of evolution. In the first phase, characterized by NCP in the ARPANET in 1970-1980 [2], the channel bit latency was small compared to the average computer buffer size. TCP became prevalent in 1980 as the bit latency increased, and approached the average file size [6]. TCP is a sliding-win-

1. This research was sponsored by the Defense Advanced Research Projects Agency through Ft. Huachuca Contract No. DABT63-91-C-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Army, the Defense Advanced Research Projects Agency, or the U.S. Government.

dow flow control mechanism for transfer of linear sequences of data, and thus works well only when the buffer space is as large as the bit latency of the channel, and where the file size is at least an order of magnitude larger than this [7]. The sliding-window mechanism in TCP (and other protocols) relies on the availability of a large linear stream of messages in order to occupy the channel during the round trip time.

Gigabit WANs are different because the channel is no longer sufficiently occupied by file transfer. A file that dominated the window size of a 1.5 Mbps channel, now occurs as a small 'blip' on a 1 Gbps channel (Figure 2). An entire file now occupies a gigabit WAN channel as a conventional RPC message occupies a 10 Mbps MAN channel.

In Figure 2, the file is 30x as large as the window size[1] of a conventional 1.5 Mbps WAN, or approximately 1 Megabit (125 K bytes). At 45 Mbps, the window needs to be 30x larger to occupy the channel during the bit latency, and so the same file no longer fills the round trip window. At 150 Mbps this effect is even more pronounced. At 1 Gbps, the file is a very small message, as compared to the round trip pipeline. If we compare the file duration to the channel latency, and graphically normalize for the larger of the two, we observe that the situation is practically reversed (Figure 3).

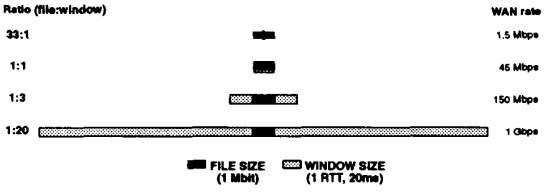

FIGURE 2. Relative sliding-window sizes (bit latency) vs. fixed 1 Megabit file size, as bit rate increases.

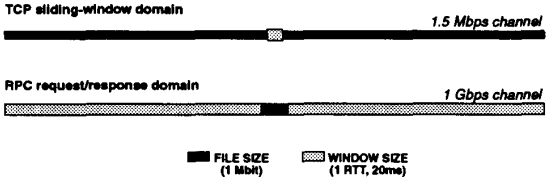

FIGURE 3. Reversal of the dominance of file duration over bit latency.

We are now in a period in which the file sizes are near the bit latency of proposed channels (at 150 Mbps rates, this is a 3 Mbit or 400 K byte file). As the channel rates increase with technological advances, the single-source channel utilization will decrease. Conventional protocols cannot otherwise sufficiently occupy the channel. These protocols were designed for low latency message exchange (RPC), or high-latency linear stream communication where the stream is much longer than the latency (TCP). We now have a case where even entire file exchanges appear as a brief message exchange, and where a channel will be idle because no data is waiting for transmission.

Satellite protocols also operate in high bit latency, high bandwidth domains. Satellite networks have unique topology and resource distribution characteristics, so their solutions are not applicable to general WAN protocols [7]. The centralized management afforded by the satellite transponder is defeated by its requisite simplicity, in order to satisfy severe weight, reliability, and power restrictions. As a result, satellite protocols use simple fixed-window synchronous protocols and a high degree of deterministic channel multiplexing to increase utilization in spite of their high latency environment.

Figure 4 shows how the size of the RAM buffers compares to the bit latency of the channel as bit rates increase; we assume here that file sizes are growing in proportion[2]. In the period between 1970 and 1985, the amount of buffer space available was 40x larger than the bit latency of the Internet. Advances in the channel rates are diminishing that gap rapidly, and buffer size will be approximately 3x the bit latency.

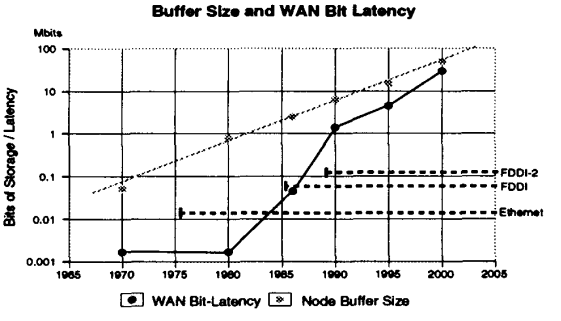

FIGURE 4. The gap between RAM buffer size and bit latency.

Figure 4 also shows the bit latencies of Ethernet[3], FDDI, and FDDI-2, based on specified maximum network sizes of 2.5 Km for Ethernet and 200 Km for FDDI [7].

1. The window size is optimally the same as the bit latency [7].

2. This is a loose bound, because we assume the entire RAM contents are dedicated to the buffer.

3. Ethernet is a trademark of the Xerox Corporation.

These protocols never incur bit latencies very close to the size of the node buffers, as will occur at T3 (45 Mbps, 1990), STS-3 (155.52 Mbps, 1995, planned), and STS-24 (1.244 Gbps, 2000, planned) rates in WANs [1].

In summary, effective channel use requires occupying the bit latency, but small transaction message sizes are a limiting factor (e.g., TCP where files are smaller than sender windows, and RPC in general).

There are two ways around this: the management of sets of files or streams, and indeterminism within a single stream. These are discussed below.

3: Imprecision of state

The main problem with existing protocols is that they are based on deterministic models of remote state. In a network with high bit latency, we cannot rely on a consistent, deterministic global state to be maintained in a timely fashion. A deterministic model will not let enough information be put into the stream to fully utilize the channel until after a reply is received.

Relaxing the model to introduce imprecision of state in the system involves partitioning the states of the model of the remote state, and increasing the message set to compensate. The result is a decrease in individual message utility. Partitioning the state requires messages to be labelled with partition identifiers, so that the message data is used only when received by its corresponding partition member state. The idea is to send "all possible next requests", but to restrict the requests to be less specific. The "set of all possible next requests" we call a *set of communication*.

For example, consider a variation of TCP's sliding-windows in which buffers are not arranged as a linear sequence (Figure 5), but rather as a tree (Figure 6). In either case, as messages are sent, the buffer space is consumed (the 'slack' in the figures), limiting the amount of information in transit. In the TCP case, the lack of 'more file to send' is an impediment to buffer use, in the high-bit latency case. In the Branching Windows case, at each point in the protocol, a decision is made about which branch to explore. As messages are received, they specify the correct remote branch, and incorrect branch components are removed. The protocol generally operates as before, except that when 'window #4' (i.e., level 4 in the tree) is active, many possible versions of that window are valid. This allows the window to expand to the bit latency, rather than being restricted by the input stream to TCP. We sacrifice precision of state, and increase the number of messages that are required, because we need to send messages to each possible remote state.

The result is that this system no longer requires feedback at each level of the tree in order to proceed and descend to the next level. We do not need to know the exact state of the remote system, so long as we keep track of ALL the possible states of that system, and communicate as if all were potentially there. Scale issues are addressed in the tradeoff between the number of partitions, and the overhead of each partition. One requirement of this method is that a particular remote node, in a single state, should not be aware that we are sending messages to his 'possible siblings'. As a result, we put conditionals, or guards, on the messages, so that the remote node receives only those messages addressed with its current state.

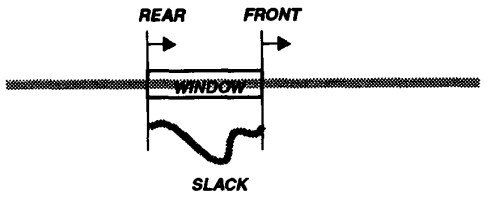

FIGURE 5. Conventional sliding-windows.

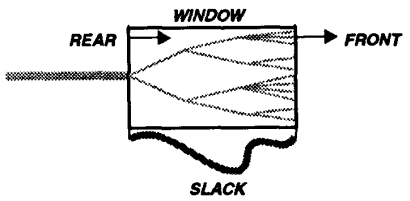

FIGURE 6. Branching windows

4: Implications

Other researchers have also examined the problems created by increased bit latency [5]. Proposed solutions use pipelining and process parallelism to increase channel utilization. Pipelining is not effective in reducing start-up latency, the kind which precedes every round-trip message exchange. Increasing the pipelining is the desired result, but cannot be achieved *per se*. The file size has become a limitation to filling the pipe. We now must consider sending multiple files or having multiple sessions to compensate.

Process parallelism uses a set of processes to fill the pipeline with messages or files. The processes' data streams are multiplexed together, so that when one process runs out of data to send, another will be activated to use the channel. The sender needs to determine the state of the receiver, because the source of the data depends on which process is currently active. Process parallelism denotes a set of collectively managed processes, but it is not the process set that permits the channel utilization, but rather their management as a set. Process parallelism is equivalent to

Parallel Communication, because the management of nondeterministic context switching is equivalent to the nondeterministic remote state in our model.

Prefetched anticipation, as used in cache management, does not help reduce bit latency induced channel underutilization. If the information could be deterministically predicted, it would have been used to fill the pipeline via conventional linear-anticipation protocols (i.e., linear lookahead, as in sliding-window). Anticipation that attempts to predict remote state, and recovers if in error, only increases the imprecision of state across the channel. Instead, a method of anticipation that uses the imprecision of state, but does not require recovery in the conventional sense, can be used effectively [9]. This is sender based anticipation, where a set of possible messages is sent, and where the anticipator's notion of remote state is refined, but not recovered.

Parallel Communication is based on sender-based anticipation, the sender using an imprecise state model of the receiver, and having the sender manage the set of processes together (rather than individually). The idea is to send the 'set of all possible next requests', and permit the local notion of remote state to become imprecise, in exchange for later refinement via received messages. Parallel Communication uses the increased bit latency of the channel to send sets of messages, only part of which are useful. The result is that the channel is used to forward messages correlated to possible futures of the receiver. This more accurately models the imprecise state of the remote participant.

One effect of Parallel Communication is that, in order to predict the possible futures of the remote node, a detailed model of the operation of the state of the remote node is required. This indicates that layering can obscure the state evolution that is required for the sender to analyze the possible states of the receiver. Parallel Communication methods are not applicable to existing protocol implementations, e.g., TCP, since they are linear only, and the bit latency exceeds the linear message size (file size). This method is applicable to the 'sets of processes' solution, because the set appears as a single (total) nondeterministic remote process (appropriately).

4.1: Implications of these observations

Thus far, we have observed that existing protocols do not permit single sources to effectively occupy a high bit latency channel. There are some obvious implications to this observation, and to the notion of sender-based anticipation as its compensation.

First, a single source cannot occupy the channel, but a set of sources can. We do not address the aggregate sharing of a channel. Such shared use is provided by either deterministic multiplexing, or nondeterministic multiplexing. Deterministic multiplexing is an agreed solution; we are concerned with cases where it is not possible. Nondeterministic multiplexing requires a higher level protocol to determine component agreement between the sender and receiver, and such a protocol is equivalent to Parallel Communication methods (as addressed later).

Second, files are messages that are too short to occupy a high bit latency channel. To make protocols regain performance, we consider the files as the component blocks of transmission, and consider the higher level structure of the file transmission. The files become the components of a rich structure of data flow, for which current protocol mechanisms are not suited.

The result of this latter observation is that existing protocols are suited to linear stream communication (i.e., conventional files), and that PC is suited to rich structure streams, such as file sets and trees. There are other examples of rich structure data streams, static examples of which are hypertext and hypermedia, and dynamic examples of which are 'object' data structures.

Another observation is that sender anticipation can also be used with linear streams (conventional files) where indeterminism is introduced by transmission errors. In this case, PC reduces to a form of forward error correction (FEC), and file components are repeated [10].

In all these cases, we are using the network as a cache, to reduce the bit latency between the sender and the receiver. This network cache differs from a conventional cache, in that its contents change over time, and this change is controlled by the source of the cache data.

5: An Example of Parallel Communication

Consider the case where the round trip bit latency, formerly much smaller than the size of the file, is now much larger (Figure 4). The channel now offers more bandwidth than conventional applications can manage. Rather than stopping at a single file or message, further messages can be sent, anticipating the requests of the receiver. This requires management of the set of messages at the sender, to prevent rollback that would nullify the latency advantage. In Figure 7, a single message sent in a long window does not utilize the additional bandwidth available. In Parallel Communication, the sender emits conditional messages in a tree, depicted here as smaller line widths and lighter lines denoting the probability of utilization at the receiver. The channel sees this tree as a conventional sequence of messages, using the entire window. The receiver in this example selects only those messages appropriate to its current state, resulting in receipt of 4x as many messages as a single-message scheme. The messages are no longer probabilistic at the receiver (solid black message segments), because its local state determines the active message component of the sent tree.

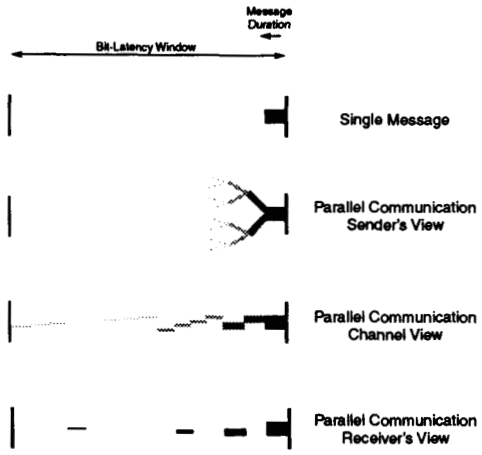

FIGURE 7. Parallel Communication visualized.

Parallel Communication is particularly well suited to hypermedia communication, where the structure of the message flow is indicated by information links. Hypermedia using interactive digital video is one example where message sizes are large (still video images or brief full-motion), and the link structure is rich and hierarchical, rather than merely linear.

5.1: A CPU-memory interface

One example that demonstrates the potential for Parallel Communication is the interaction between a workstation and its disk server. The workstation consists of the CPU, RAM, and local I/O (user, data measurement, etc.); the server contains the read-only program. The server is the receiver of the opcode stream, and the workstation is the sender. The state of the receiver is the CPU program counter. We present this interface in terms of an opcode stream, although it can be applied to page-level communication as well.

In conventional protocol interaction, the server responds to explicit requests from the workstation. Each request indicates the receiver's current program counter, and the sender reacts by emitting the appropriate opcodes. Nothing occurs in the latency between requests.

In Parallel Communication, the server models the workstation as a set of possible program counters. The server emits an opcode appropriate for each of the possible program counter during the interval between requests. As a result, requests are preemptively serviced. The workstation receives an early reply to a request, compared to the actual latency interval. The latency is thus reduced.

This method works only where the interaction is latency limited, rather than bandwidth limited.The server's model of the workstation's state is constructed through opcode analysis at the server, which is computationally intensive. We also assume that the receiver can process the opcodes at the rate received; if not, some sender pacing or receiver buffering is required to prevent overrun during some parts of the communication, particularly when the message guards cannot be computed at the rate received.

In the case where the processing power and bandwidth are available to compensate for latency, speedups of 3000x are possible [8].

6: Future directions

We need to apply this technique to higher levels of the protocol, because that is where the information resides, i.e., information on the characteristics of the sets of processes. For example, we need an interface in which the conventional application-layer OUT signals of OPEN, CLOSE, and SEND, and the IN signals of OPEN_REQUEST, CLOSE_REQUEST, and RECEIVE, are supplemented by the OUT signal of SEND_CONDITIONAL_REQUEST, and IN by RECEIVE_CONDITIONAL. In this way the protocol stack can indicate to the application that additional bandwidth is available for Parallel Communication, and the application can indicate sufficient remote state so that only applicable messages are passed up to the remote receiver.

In existing protocols (IEEE 802.2 LLC [4], TCP [6]), there is a set of flow control signals that is used to negotiate a description of the 'lossless' buffering available for the channel. This buffering currently represents a capability of the buffer software, although that capability is sometimes determined as a function of round trip capacity measurements on the data link. The network layer at the source node can receive a signal from its local transport layer indicating the amount of data that layer can accept for 'lossless' transmission. The destination node can send a signal to its local transport layer indicating the amount of data that the destination can accept without overflow. These signals model a notion of the stream as a linear sequence of messages that will occupy these buffers. The buffer sizes indicated are restricted by a set of rules, i.e., that the source has enough space to occupy the round trip bit latency, that the receiver has the same space, and that the file is much smaller than these buffers.

Existing interfaces do not consider the case where the source is more severely limited by the messages sent than by the channel or receiver capabilities. We are currently developing the extensions to TCP providing the required signals for Parallel Communication. The control interface augments the conventional signals of OPEN, CLOSE,

SEND, and RECEIVE with signals to the application layer that request conditional data from the sender, and request state resolution information from the receiver. This effectively splits the linear buffer sequence into a tree of possible remote buffer states. The remote buffer remains a deterministic linear sequence; only the sender's perception of that sequence changes to accommodate latency-induced imprecision. The sender's buffer needs information from the application layer describing the ways in which the remote state can become less precise over time, and ways to specify labels to guard the receipt of the messages. The receiver's buffer management is augmented to use information about its local state to filter the incoming data stream, and restrict receipt of messages.

6.1: Details of NFS Parallel Communication

The required TCP extensions support branching windows and provide application-layer signalling of excess bandwidth that can be used for latency reduction. These extensions support the development of application-layer modifications, which can be implemented as intermediaries to existing protocols (Figure 8). Here we replace the conventional RCP/UDP NFS interface with a branching-TCP version.

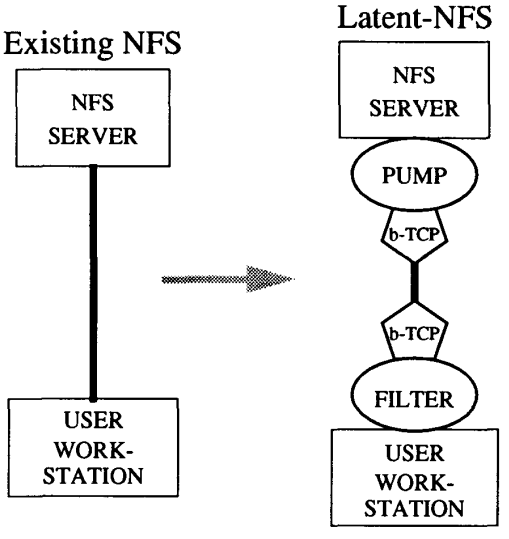

FIGURE 8. Implementation of the NFS intermediaries called the *pump* and *filter*.

The *pump* and *filter* implement the application modifications. The pump modifies the server side of the interface; the filter modifies the client side. The pump manages the sending of all possible next requests, and manages the possible states of the client. The pump uses the server-side TCP signal of excess bandwidth to initiate sending the anticipatory messages, and the branching window allows the pump to send alternate streams of messages to the client. As the pump emits these messages, the branching in the server-side TCP increases.

The filter allows the client application to receive only those messages that correspond to a particular state. The client application signals its TCP interface to filter the incoming stream based on a branch identifier. This client-side TCP also indicates branch selections to the server-side TCP, to reduce the branching at the server.

The branching increases by the pump and decreases by the filter's messages to the pump determine the extent to which the excess bandwidth can be used to reduce the observable latency [9].

General algorithms for the pump and filter are given in detail in [9]. A specific pump and filter for managing CPU-memory opcode communication is given in detail in [8]. The pump and filter shown here (Figure 9) exemplify characteristics of the TCP branching windows mechanism and the application-layer extensions together.

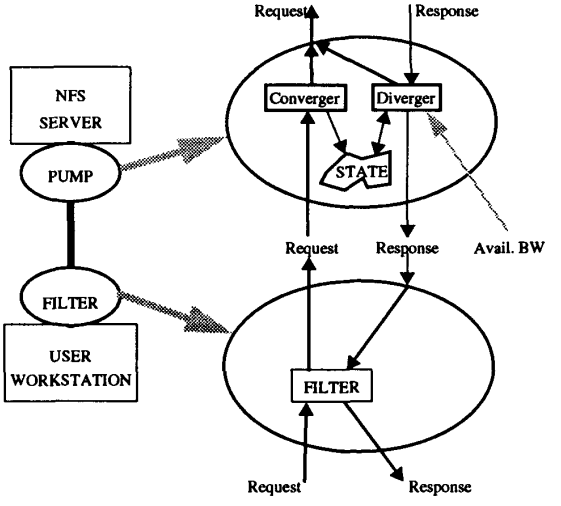

FIGURE 9. Detail of the interaction of the *pump* and *filter*.

6.2: Some preliminary performance results

In earlier work, we performed some experiments to measure the potential performance increases through Parallel Communication methods. Using a simplified model, and some dynamic opcode traces, we were able to compute the expected benefits.

For these observations, we define speedup as the ratio of actual execution time to "optimal" execution time, the latter being the time to execute on a local system with zero

latency and infinite bandwidth between the CPU and memory (Equation 1). For these equations, we define:

N = number of pages fetched
t = time to execute a page
missrate = percentage of pages not in the cache
pBW = page bandwidth
pD = page degree
pL = page linearity
rtt = round trip time

The page bandwidth is the number of pages communicated per unit time over a channel, the page degree is the average number of pages that a single page ever jumps to, and the page linearity is the average number of pages executed in a linear sequence (due to 'running off the end').

$$OPTIMALtime = N \times t \quad \text{(EQ 1)}$$

Equation 2 describes the execution time when the CPU is separated from the memory. Equation 3 describes the same when a page cache is added near the CPU, and the miss ratio is measured over the trace.

$$DISTANTtime = N \times (t + rtt) \quad \text{(EQ 2)}$$

$$CACHEDtime = N \times (t + rtt \times missratio) \quad \text{(EQ 3)}$$

We can approximate the average expected execution time achieved by using Parallel Communication by the formula in Equation 4 [8].

$$PCtime = N\left(t + \left(1 - \frac{t \times \log\left(1 + \frac{rtt \times pBW \times (pD-1)}{pL \times pD}\right)}{pD \times rtt}\right)\right) \quad \text{(EQ 4)}$$

We have already performed such measurements at the individual opcode level on the Sun SPARC architecture running the GNU C compiler weighted benchmark from the SPEC Benchmark Release 1.0 [10], a TEX benchmark noted in [3][1], and C-language versions of both Dhrystone and Linpack[2]. Our results are equivalent to an NFS trace where the page size is a single opcode. Although these conclusions do not imply correspondence with NFS case, they have justified further research including the NFS experiments.

1. The software supplement of [3] is available via anonymous FTP at max.stanford.edu.

2. These benchmarks are available via Internet e-mail; send queries to netlibd@surfer.epm.ornl.gov.

Figure 10 shows the channel utilization curve, for conventional (solid line in the figure) and versions of Parallel Communication (dashed lines). The channel utilization for the conventional case is determined by the channel occupancy; the PC case is determined by the occupancy divided by the expected utilization. PC uses the channel to send messages, some of which are not used when received. We have plotted utilization as 'utilized messages per unit time'. The raw channel utilization in the PC cases is always 100%, because possible future requests are always being sent. The different PC versions denote different versions of implementation, some less complete than others.

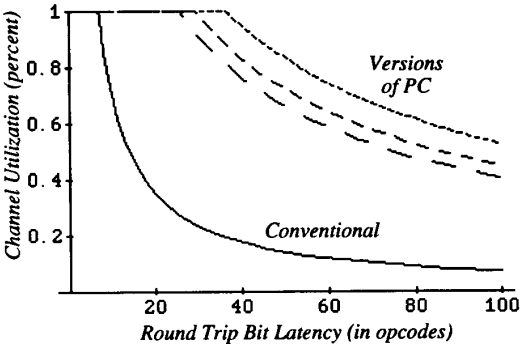

FIGURE 10. Effective channel utilization

The expected speedup of a program execution is defined as the ratio of the PC execution time (dashed lines in the figure) divided by the remote execution time (solid line) (Figure 11). As the bit latency increases, PC exhibits a linear increase in utilization for a brief time, then a logarithmic increase.

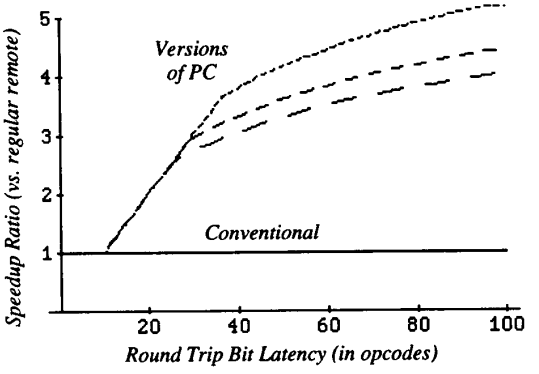

FIGURE 11. Expected execution time speedups of various PC implementations.

6.3: A note on beating the setup latency

Earlier we noted that all measurements and comparisons to existing protocols were made midstream, whereas setup round trip latencies were ignored.

One way to address channel latency during the setup exchange is to apply PC methods to the setup itself, rather than just to the transport portion of the protocol, as has been done here. The initiator of a connection sends an 'open request' as usual, and starts sending data in anticipation of the future reception of the 'connection accept' message, thus avoiding the initial round trip setup latency.The data can be used only in a conditional (revocable) sense, either by rollback or branching histories. If the request is denied, the data sent must have been ignored by the receiver.

The result is equivalent to a fast-setup, where the data transport occurs before the setup completes. This is accomplished without modification to the protocol; by implementing the protocol using PC methods, a degree of asynchronous setup is achieved.

7: Conclusions

We have to date developed the description of a possible channel utilization protocol method, and performed some preliminary measurements that indicate a potential gain. We are in the process of developing a full emulation experiment, in order to measure the gains on an NFS system at the page level. A final implementation of the system, pending the outcome of these measurements, is also planned.

8: Acknowledgments

We would like to thank Ted Faber of the Univ. of Wisconsin, Madison, and Jon Postel, Bob Felderman, Steve Casner, Eve Schooler, and Greg Finn of ISI for editing and feedback on the content of this document.

9: References

[1] ANSI - American National Standards Institute, *Digital Hierarchy Optical Interface Rates and Formats Specification*, T1.105-1988, Draft March 10, 1988.

[2] Carr, C.S., Crocker, S.D., and Cerf, V.G., "HOST-HOST Communication Protocol in the ARPA Network." In *Spring Joint Computer Conference*, AFIPS, 1970, pp. 589-597.

[3] Hennessy, John L., and Patterson, David A., *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Mateo, CA, 1990.

[4] Institute of Electrical and Electronics Engineers, *Logical Link Control*, American National Standards ANSI/IEEE STD 802.2, 1985.

[5] Kleinrock, Leonard, "The Latency / Bandwidth Tradeoff in Gigabit Networks," *IEEE Communications Magazine*, Vol. 30, No. 4, April 1992, pp. 36-40.

[6] Postel, Jon, "Transmission Control Protocol," RFC-768, DARPA Network Working Group Report, USC / Information Sciences Institute, August 1980.

[7] Tannenbaum, Andrew S., *Computer Networks*, Prentice-Hall, NJ, 1988.

[8] Touch, Joseph D., and Farber, David J., *An Active Instruction Decoding Processor-Memory Interface*, patent pending, Univ. of Pennsylvania, September 1991.

[9] Touch, Joseph D., *Mirage: A Model for Latency in Communication*, Ph.D. dissertation, Dept. of Computer and Information Science, Univ. of Pennsylvania, 1992. Also available as Dept. of CIS Tech. Report MS-CIS-92-42 / DSL-11.

[10] *The SPEC Benchmark Report*, Waterside Associates, Fremont, CA, January 1990.