Protocol Parallelization

Joseph D. Touch[*]

USC / Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA, 90292-6695, U.S.A., (touch@isi.edu)

Abstract
There is increasing concern about the capability of existing protocols to keep pace with communication rates, as rates approach the gigabit range. This assumes a sort of "protocol bottleneck." Many similar bottlenecks are alleviated by the use of parallelism, so one hypothesis is to "parallelize" protocols. We examine the pros and cons of this hypothesis, and the dimensions to which parallelism might be applied. We distinguish the unique communication issues that result. Our conclusions indicate that conventional parallelism may not be applicable to protocols. New types of parallelism become more significant in this light. These include information parallelism (Parallel Communication) and packet-train parallelism.

Keyword Codes: C.2.2, C.2.4, C.5.0
Keywords: Network Protocols, Distributed Systems, Computer System Implementation

1. INTRODUCTION

High-speed networks have brought renewed emphasis on protocol performance optimization. Some optimizations focus on machine-specific implementation tuning [17], or protocol-specific implementation tuning [10]. Others have proposed "RISC" protocols, removing cumbersome functions from the protocol itself (XTP [4], VMTP [3], etc.). A number of projects have observed that bottlenecks in other disciplines are often alleviated by applying parallelism, either spatial or temporal (i.e., pipelining). Here we investigate the kinds of protocol parallelism possible, summarize the current attempts at parallelization, and make observations about their feasibility and limitations. We also observe some unconventional types of parallelism, such as information parallelism, which may be viable. We also consider whether the parallelization addresses protocol speed or latency.

First we define protocol parallelization, and what it requires. We present a framework in which to compare parallelization methods. Then we consider prior work regarding protocol optimizations in general, and with respect to parallelization in specific. Finally, we make some observations about gaps in the design space, and novel protocol methods which suggest types of parallelism not conven-

[*]. This research was partially sponsored by the 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 Advanced Research Projects Agency, or the U.S. Government.

tionally considered. We conclude that protocol processing itself may not benefit greatly from parallelization, and that parallelizing the control and feedback may prove beneficial in increasing channel utilization and reducing latency.

2. WHAT'S THE PROBLEM?

Even though the "communications bottleneck" is an observed phenomenon, its cause isn't well understood. There is a substantial difference between the bandwidth of a workstation backplane (typically near 1 Gbps), and the rate at which communication can occur to an external network (typically near 33 Mbps). The difference between these rates is a function of many factors, that include the costs of *asking* for state information, *thinking* about the question and forming a reply, and *answering* by sending the reply, i.e.,:

- *processing*
  (ability to think fast)
- *bandwidth within and between hosts*
  (ability to feed the questions to the thinker)
- *sourcing limits*
  (ability to have enough questions to think about)

Thinking is a processing bottleneck, exhibited in the processing speed of TCP/IP and the operating system (OS) interface involved in the transaction. The processing bottleneck for TCP has been addressed by parallelism [16], [1] (both discussed later), even though its performance has been shown not to be the predominant limitation to communication [17], [10]. Other components of the protocol stack have also been parallelized, e.g., via pipelining of the IP check-sum with the data transfer [5]. Processing in the OS has also been considered, although most of the focus has been on data transfer issues we denote as *answering* bottlenecks.

Answering is a data transport bottleneck, i.e., bandwidth limitations. This is currently being addressed by parallelizing the internal data paths of workstations. The basic idea is to replace the internal bus with a more general topology. The Cambridge Desk Area Network (DAN) [9], and ISI's NetStation [7] are examples.

One interesting question assumes nearly instantaneous *thinking* (protocol processing) and *answering* (bandwidth), and asks, "is there anything else?" Presume that TCP has infinite window sizes, and can run at 800 Mbps (it can on a CRAY [17]). Presume that we have a NetStation, in which each component of a workstation (disk, RAM, display, etc., [7]) can both source and sink at 800 Mbps. What then?

This is the question of *asking*. Latency is the final bottleneck [19]. Ultimately, answers can be given only as fast as questions arrive. If the next question depends on the current answer, round trip propagation latency is incurred between *asking* rounds. Assuming *thinking* and *answering* are not the bottlenecks implies *asking* is.

In terms of current protocols, even with nearly infinite bandwidth and processing, TCP can "fill the pipe" only so far as the source data (answer) exists. Once an entire data stream is sent, nothing more is communicated until the next query arrives. It is here that we believe parallelism is best applied, to the parallelization of possible next questions and thus answers. We call this Parallel Communication, and it is based on parallelization of the information stream [19], [21].

3. PARALLELIZATION ISSUES

Parallelization uses replicates to perform the work of a single entity, and involves considering replication dimension, mapping function, scale limitations, replicate interference, overhead, and expected gain. The type of entity replicated is the replication dimension. Common protocol dimensions include per-protocol, per-connection, per-packet, per-layer, and per-protocol function. This subsumes the difference between spatial and temporal (pipelining) parallelism, because temporal parallelism is spatial parallelism with head-to-tail interdependence between components, and usually implemented per function. From this point, we therefore do not consider pipelining as a distinct case.

Other dimensions not commonly considered include per-packet train, and per-information stream. Packet trains are sequences of packets that act as a unit in the protocol; a common example is a fragmentation group. Information streams are alternate packet sequences, such as would occur with breadth-first (BFS) source anticipation [21]. BFS source anticipation is parallelizing possible future questions and their answers, in order to keep the communications mechanism occupied between actual questions. Dimensions not considered are per-host address (issue for routers only), and per-application (equivalent to per-protocol).

The mapping function helps determine whether a replication dimension is feasible. Given a dimension, the map indicates how incoming data is switched to the appropriate replicate. Some dimensions are easy to map (per-protocol, per-layer, per-connection). Others require almost as much effort to map as would be required to emulate the protocol (per-protocol function).

Scale limitations indicate the bounds on the expected replication growth. If choose per layer, we are commonly limited to seven replicates (OSI model), or thereabouts. Per protocol function is often limited to five (TCP, TP-4), because the protocol definitions are often described as mostly serial processes.

Replicate interference describes the interaction required between the parallel components, and whether the work is increased as a result. Per-connection replicates are usually defined as not interacting, but redundant operations in protocol layers often implies that per-protocol and per-layer replicates interfere. Per-packet replicates interfere heavily, because they affect common connection and protocol parameters (i.e., state information).

Overhead includes the cost of replicate interference, as well as the general overhead of switching according to the mapping function, process creation and removal (if dynamic), and other costs of a parallel implementation.

Expected gain measures the overhead costs with the scale limitations, to determine whether the parallelization proposed is feasible or effective. Some types are feasible but not effective, because replication is easy and overhead is low (per-protocol), but the expected gain is minimal. Other types are neither effective nor feasible (per byte!), because the overhead outweighs any expected gain, or is at least as large as the protocol itself.

For each parameter, we consider whether a real bottleneck is being addressed Recall that TCP can run at 800 Mbps [17], however not across a real link. Is the limitation the processing, or the sourced information itself? Is speed a real issue, or is latency? Before we continue, we should reevaluate what the existing bottlenecks of a "heavyweight" transport protocol are.

Our conclusions are summarized in Table 2, near the conclusion of this paper.

3.1 Existing bottlenecks

Much work has been done to address existing protocol processing bottlenecks. This includes *thinking* optimizations, such as header processing optimizations, state optimizations, implementation optimizations, and augmenting the general processing power of the system. It also includes *answering* optimizations, including data transport path optimizations.

Header processing optimizations include "fast-path" optimizations and header prediction [10] are specific instances of general cross-product protocol optimizations such as Protocol Bypass [24]. The technique takes the cross product of all protocol functions and layers, factors out the statistically favored states and implements them as special (fast) cases. The remainder of the protocol is implemented as before.

State optimizations result in RISC-style 'lightweight' protocols. They implement the cross-product protocols (as above), and remove the non-favored states from the protocol. Examples include VMTP [3] and XTP [4].

Implementation optimizations include code tweaking such as has been done on the Cray TCP [17], and generic TCP [10], as well as *Integrated Layer Processing* (ILP) [5]. They also include optimizing the underlying OS interfaces, such as Jacobson's fast-sockets, and the x-Kernel optimizations [18].

Other optimizations address general processing issues, and would benefit conventional applications as well a protocol processing. These include fast context switching and hardware header processing. The latter is a flavor of support processor, other examples of which include FPUs, string processors, and graphic engines.

Protocols exhibits data transport limitations even after avoiding multiple data movement. This is evidence of conflict between the topology of the external network, and the internal backplane communication, an *answering* bottleneck. Proposed solutions involve "moving the network into the backplane", such as in the Cambridge Desktop Area Network (DAN) [9] or ISI's NetStation [7].

There are other limitations, *asking* bottlenecks, as have been recently observed [21]. In a high bandwidth-delay product network, the data source itself becomes a limit to the channel utilization (not enough questions). Assuming the window size limits of the current TCP specification are fixed, what will fill these windows? Measurements indicate that network bandwidth-delay products are increasing at a faster rate than that of the end system, so it's not clear that file sizes will increase in proportion to network rates, such that a larger window will be usable [22].

4. PRIOR WORK

Optimized TCP focused on increasing the window size to accommodate higher bandwidth-delay products, so-called "long delay" [11], "high-speed" [12], and "high-performance" [13] TCP. The large windows permitted flow control feedback to occur over byte blocks, rather than bytes themselves, permitting longer *answers*.

"High-performance" TCP was specifically designed for high bandwidth-delay environments, observing that the bandwidth-delay product was the issue, rather than the latency or the speed (as in prior proposals) [13]. A critical response indicated that some of these proposed optimizations were harmful, and suggested flow control occur over records, rather than bytes. This indicates a message structure beyond that normally assumed for TCP. A current proposal augments TCP to handle transactions [2], to accommodate some of the record-structure suggested before. This line of research is realizing that bandwidth-delay products are the primary issue (i.e., a question bottleneck), and that large windows don't solve the problem (because they solve answering). Additional structure in the information stream is required, beginning with a record structure

There are other proposals that observe the limitation of the exchange of state in a high bandwidth-delay product environment, and attempt to accommodate this.

Periodic exchange of state as a protocol mechanism is the basis of the Delta-t protocol [23], and has also been mentioned in TP++ [6] and the SNR "leaky-bucket" protocol [15].

A recent proposal indicated that high bandwidth-delay product environments change bandwidth-bound systems to be latency- or server-bound [14]. The sources become message limited (question-bound), and the channel utilization changes only if the message sizes increase. The proposal suggests that multiplexing will increase channel utilization, just as multiprocessing increases processor utilization in the presence of I/O communication latency.

A reply to this proposal observed that multiplexing would not so much solve the problem as define it away [20]. Deterministic multiplexing is equivalent to lower bandwidth-delay product links. The real issue is protocol state imprecision, induced by high bandwidth-delay products and variance in protocol function (nondeterministic protocols) or in latency (delay variance). Nondeterministic multiplexing just pushes the state imprecision problem to the multiplexer control level. The only known solution uses message stream structure (records, branches, and recursion) to permit source anticipation of remote state [19]. The message stream carries responses to "parallel futures" of the remote state. One application which supplies the requisite structure is distributed hypermedia, such as in the World-Wide Web.

5. CURRENT WORK

Recent work attempts to parallelize the processing of particular protocols. These include per-packet and per-function dimensions, both empirically and analytically derived. The results have been disappointing thus far, with a scale limit of 5 parallel processors for most experiments.

Early work tried per- layer parallelism, a sort of data flow architecture [25]. They decomposed TP-4 onto Transputers, where layers 3 and 4 were each assigned one processor for each of send and receive functions. The result was 4-way parallelism, with no further decomposition proposed.They concluded that the protocol definitions were not conducive to further parallel decomposition, especially because of the overlap of functions between layers.

Other work focused on per-function decomposition of OSI protocols [8]. They considered TP-4 and 802.2 LLC protocol decomposition. They decompose their protocols per-layer 4-ways, by allocating 2 processors each for send and receive. Further decomposition follows a dataflow diagram of the protocol, where functions are divided among the two processors. They observe that the interaction between the two processors on each side has significant overhead, and that further decomposition is not feasible because of increased communication costs. When they attempted to apply their technique to TCP/IP, they found a scale limit of 4 processors there as well. Some of their results may be an artifact of the communication topology of Transputers, which are 4-way connected.

More recent attempts factored TCP/IP as if its functional components were executing on a statically load-balanced compiler [16]. At first, they partitioned TCP into three functional components - send, receive, and timer. Their analysis concluded that TCP could be parallelized further by distinguishing between data and control, such that there were four components - *send_data*, *receive_data*, *control*, and *timer*. They found that the bulk of complexity lies in the control, which isn't surprising (this is the essence of a protocol). By making additional assumptions (not necessarily valid ones), they could reduce the work of the control protocol.

These simplifications include assuming a stable round-trip time (thus removing it from the protocol state), thus assuming a stable window size (also removing it from the state), and by performing control functions infrequently (not per-packet).

Unfortunately, this changes the operational semantics of TCP, and it's not clear whether their implementation will interoperate with standard TCPs. We observe that this is just the "big packet" solution to protocol optimization, such as in NET-BLT (long answers). Low bandwidth protocols act like high bandwidth periodic protocols (Figure 1). High bandwidth eager-delivery (as soon as possible) protocols increase the rate of headers, thus the effort of protocol processing. Rate-based flow control reestablishes the header rate, slowing the protocol processing down (Table 1). The authors note that rate policing is required for their solutions to avoid the increase in header rate.

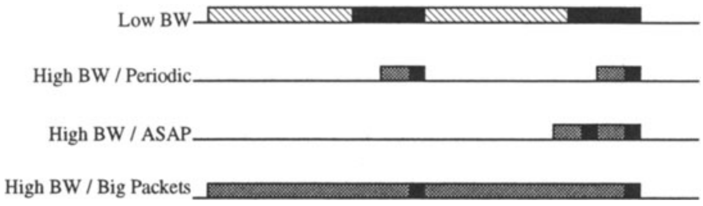

Figure 1.
Equivalence of types of protocols
- the top two are equivalent, the bottom two are not.

Increasing the packet size (as in NETBLT), or, equivalently, spacing the "real" headers out (ones that affect protocol state), results in large-packet protocols, which act like their more sluggish counterparts. A protocol is sensitive to the header processing rate. Allowing larger packets or fewer control headers is equivalent to assuming the stability of the slower protocol system. Large packets work only where state is a function of clock-time, not the bandwidth-delay product.

Table 1
Big packets are really slow/sluggish protocols

BW	rule	hdr/time	data/time	hdr/data
low bw	-	low	low	avg
high bw	periodic	low	low	avg
high bw	ASAP	high	high	avg
high bw	huge packets	low	high	low

They also observe that the parallelism is affected by state imprecision. Such imprecision is characterized by the number of possible vs. actual headers outstanding (i.e., effects of latency on imprecision), as well as the number of actual controls vs. data between the controls (effects of transmission and reception on imprecision).

Others considered per-packet parallelization [1]. They measured parallelism using simulations, and implementations of a multiprocessor x-Kernel implementation with spin-locks. They observed that the parallel processing contends for the shared protocol state (the Connection Control Block). They claim TCP saturates at a parallelism of 7 (measured as 5) with a speedup below 5x (measured as 3.5x), but that UDP didn't appear to saturate (through N=20). This is because TCP has significant shared state (the sliding window flow control), but UDP does not. We note that this per-packet experiment reaches a similar parallelism limit as per-function.

Other results were reported at the CNRI Gigabit Workshop in 1993. V. Jacobson reported that his fast TCP relies on state transition effects, rather than the full RFC-specification of the protocol. Inbound traffic processing was the bottleneck, especially where demultiplexing isn't stable; this corroborates our conclusion that Kleinrock's multiplexing solution isn't sufficient. The Nectar project tried a partitioned send/receive TCP, and encountered problems with shared CCB access as well. The IBM group reported on general parallelization using pipelining, and found that the component coupling was tight for TCP, and loose for IP. Other projects (HP/Witless, LANL CASA) examined using big packets to optimize throughput (as large as 4 Kbyte). They rely on the stability of the protocol and environment, as discussed before.

6. OBSERVATIONS

At this point we have several observations to make. First, parallelization per-packet or per-function does not help much. These results address some performance issues, but do not scale as the bandwidth, number of headers (i.e., packets), or available processors increases. As a result, they are of debatable utility. Other dimensions, including per protocol, per connection, per application, etc., are also of debatable utility, because we assume a required solution will provide high throughput for a single connection of a single protocol to a single application (1 Gbps to the user application).

These observations are important because they imply that *thinking* parallelism does not help protocol performance. *Answering* (bandwidth) bottlenecks are disappearing, as we remove simplifications (i.e. bus backplanes) from our workstation designs, and move towards a networked-backplane, thus parallelizing the *answers*. When these are solved, only one bottleneck remains - latency, i.e., a not enough *questions*.

We also made other observations. First, a protocol doesn't know what time it is, only how many bits are in transit. Consider slowing down existing hardware - from the CPU through the communications media interface. The protocol will work exactly as before. Even protocols with absolute clock timers will still function, because the clock time will be equivalently slowed.

So the rate issue is a red herring with respect to protocol processing. A protocol does, however, know how many bits are in transit, both in terms of the amount of state required, and the effort required to manage that state. Protocols are bandwidth-delay product sensitive [19]. Further, many existing parallelization techniques do not address bandwidth-delay product (the *question* bottleneck). One exception is [16], but their conclusion is that in order to avoid the problems that bandwidth-delay product causes, they exclude the high-frequency header domain.

In our work on an abstract model of communication latency (Mirage), and its protocol instantiation (Parallel Communication), we found a more direct correlation between header frequency, action, state, and stability. We conclude that one area of parallelism worth examining might be information parallelism (parallel *questioning*), where alternate streams of anticipatory data are present at the source.

Another area of consideration is the packet train. Packet trains are created when a large packet is fragmented into many smaller lower-layer packets. The control occurs at the head and tail of the train, and within the train. As a result, the difference between the intra-train state and inter-train state is well defined, and might be a useful dimension to parallelize. However, the benefits apply only to the segmentation and reassembly protocols.

6.1 LISN/MISN/WISN

We observed in our research that area isn't the issue, i.e., LAN, WAN, MAN. We find that protocols operate differently depending upon the information separation (IS), measured as bits in transit (bandwidth-delay product). That is, bandwidth-delay product is the formula, and information separation is what it describes.

So LAN/MAN/WAN become LISN/MISN/WISN. We define these distinctions based on the comparison between the messages sent and the IS. In LISN, the message is much larger than the IS, so existing sliding-windows protocols suffice. In MISN, the message is about the same size as the IS; this corresponds to RPC or remote evaluation (REV) domains. WISN requires Mirage / Parallel Communication to utilize the channel effectively, because the inter-message effects dominate, rather than intra-message. This is where structure in the information stream is used.

6.2 The protocol solution space has holes

When we considered the protocol solution space based on the IS, we found some holes in it (Figure 2). We could reduce the separation, or deal with it. We can remove the IS by moving the data to the code (RPC) or the code to the data (REV).

Alternately, we can deal with the IS directly. If the data is organized as a long linear stream, and the stream is longer than the IS (i.e., the LISN domain), we use sliding windows. Otherwise, we need to accommodate more complicated structures such as branching and recursion of the state space, to emulate the imprecision of state induced by the latency.

Examples of structured streams are hypermedia (interactive media), and reactive / proactive control systems. The resulting system has parallelism of action, and permits the state spaces to be partitioned while the protocol is running. This permits dynamic load balancing of the resulting processes among whatever set of processors area available.

As a result, Parallel Communication directly addresses the WISN domain, and scales to use additional processors as the bandwidth-delay product increases. The interaction between the partitions is well defined by the Mirage model. The source anticipation describes process factoring, and receiver feedback terminates source processes whose anticipation is not needed.

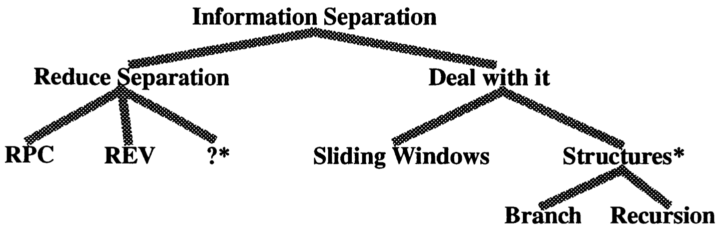

Figure 2.
Holes in the protocol state space (starred {*})

7. INFORMATION PARALLELISM

As we have discussed, the high information separation is the cause of inefficiency in the use of the communication stream. Parallelizing the protocol won't help solve this, because conventional protocols "run out of stuff to fill the pipe with". Information parallelism is a way to fill the pipe.

The protocol we use for information parallelism is called Parallel Communication. Simply put, it is sender-based preloading of a cache – in this case, the cache is the bandwidth-delay product (the information separation). The sender is sending preemptive replies to expected requests from the receiver. Each of these preemptive replies accommodates some possible state of the receiver; the set of replies represents a parallelism in the information between the sender and receiver.

A protocol for sender-based cache preloading isn't difficult, but when recursion of receiver state is considered, the mechanism becomes complex. We have developed such a mechanism for Parallel Communication.

We are currently proposing to use Parallel Communication to augment the WWW client/server (browser/server) interaction, as in Figure 3. The goal is to reduce the user-perceived response time, and permit the WWW browsers to act more like remote user-interfaces than the sluggish transaction-based mechanisms they currently are.

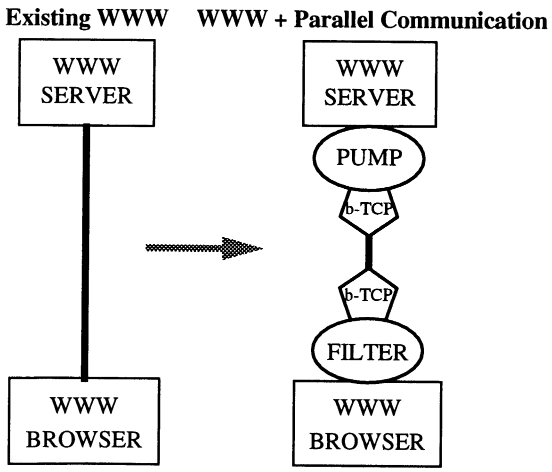

Figure 3.
WWW server and browser augmentation for Parallel Communication

For the WWW, the way in which information parallelism is determined is to filter the HTML (hypertext mark-up language) for links to other files. The other files are send as a set of information-parallel replies to the 'buttons' on that page.

8. CONCLUSIONS

We conclude that parallelizing protocols doesn't necessarily require parallel processing of existing state or packet data. It may be more useful to parallelize the possible states of the other side of the link, in order to scale as bandwidth-delay products increase. Our particular observations are summarized in Table 2.

Table 1
Summary of conclusions

DIM.	MAP	SCALE LIM.	REPL. INTFR.	OVRHD.	GAIN
protocol	easy	Y(1)	N	low	low
connection	easy	Y(1)	N	high	low
packet	easy	Y(5)	Y	high	low
application	easy	Y(1)	N	low	low
packet train	hard	Y(<10)	some	low	med.
function	none!	Y(5)	Y	high	low
layer	easy	Y(7)	N	low	low
information	moderate	N	some	some	high

We observe that other optimizations, such as big packets or periodic control, only require further assumptions about the stability of the state and state evolution. They do not address the protocol performance issue; rather, they attempt to make the protocol disappear.

Per-packet or per-function parallelization hits state precision limits, and does not scale beyond a factor of 5 or so. The better solution is to determine and support further structure in the communication stream, and the parallelism of possible future states that affords.

9. ACKNOWEDGEMENTS

This paper is the result of an ongoing discussion at ISI regarding the possibilities of parallelization of protocols. Herb Schorr proposed the question at ISI, and the members of the HPCC Division provided ample feedback. We especially appreciate the efforts of Jon Postel and Paul Mockapetris, both of ISI. Carolyn Nguyen of AT&T also provided feedback on the debate.

REFERENCES

[1] Bjorkman, M., and Gunningberg, P., "Locking Effects in Multiprocessor Implementation of Protocols," *Proc. ACM Sigcomm*, Oct. 1993, pp. 74-83.

[2] Braden, R., "Extending TCP for Transactions -- Concepts," Network Working Group RFC-1379, USC/ISI, Nov. 1992.

[3] Cheriton, D.R., "VMTP: A Transport Protocol for the Next Generation of Communication Systems," *Computer Communication Review*, Aug. 1986, pp. 406-415.

[4] Chesson, G., et. al., *XTP Protocol Definition,* Protocol Engines, Inc., Dec. 1988.

[5] Clark, D., and Tennenhouse, D., "Architectural Considerations for a New Generation of Protocols," *Proc. ACM Sigcomm*, Sept. 1990, pp. 200-208.

[6] Feldermeier, D., "Comparison of Error Control Protocols for High Bandwidth-Delay Product Networks," In participants proceedings, IFIP Workshop on Protocols for High Speed Networks, Nov. 1990.

[7] Finn, G., "An Integration of Network Communication with Workstation Architecture," *ACM Computer Communication Review*, V. 21, No. 5, Oct. 1991.

[8] Giarrizzo, D., Kaiserswerth, M., Wicki, T., and Williamson, R., "High-Speed Parallel Protocol Implementation," in *Protocols for High Speed Networks*, Rudin, H. and Williamson, R. Eds., Elsevier, 1989, pp. 165-180.

[9] Hayter, M., and McAuley, D., "The Desk Area Network," *ACM Transactions on Operating Systems*, Oct. 1991, pp. 14-21.

[10] Jacobson, V., "Congestion Avoidance and Control," *ACM Computer Communication Review*, Oct. 1988, pp. 314-329.

[11] Jacobson, V., and Braden, R., "TCP Extensions for Long-Delay Paths," Network Working Group RFC-1072, LBL and USC/Information Sciences Institute, Oct. 1988.

[12] Jacobson, V., Braden, R., and Zhang, L., "TCP Extensions for High-Speed Paths," Network Working Group RFC-1185, LBL and USC/ISI, Oct. 1990.

[13] Jacobson, V., Braden, R., and Borman, D., "TCP Extensions for High Performance," Network Working Group RFC-1323, LBL, USC/ISI, and Cray Research, May 1992.*IEEE Communications Magazine*, Vol. 30, No. 4, April 1992, pp. 36-40.

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

[15] Netravali, Arun N., Roome, W.D., and Sabnani, K., "Design and Implementation of a High-Speed Transport Protocol." *IEEE Transactions on Communications V. 38,* N.11 (Nov. 1990), pp. 2010-2024.

[16] Nguyen, C. and Schwartz, M., "Reducing the Complexities of TCP for a High Speed Networking Environment," *Proc. IEEE Infocom*, Mar. 1993, pp. 1162-1169.

[17] Nicholson, A., Golio, J., Borman, D.A., Young, J., and Roiger, W., "High Speed Networking at Cray Research," *ACM Computer Communication Review*, V. 21, N. 1, Jan. 1991, pp. 99-110.

[18] Peterson, L., Hutchinson, N., et. al., "The x-Kernel: A Platform for Accessing Internet Resources," *IEEE Computer*, V. 23, N. 5, May 1990, pp. 23-33.

[19] 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.

[20] Touch, J.D., and Farber, D.J., "Reducing Latency in Communication," letter to the editor in *IEEE Communications Magazine*, Feb. 1993, pp. 8-9.

[21] Touch, J.D., "Parallel Communication," *Proc. IEEE Infocom*, Mar. 1993, pp. 505-512.

[22] Touch, J.D., and Farber, D.J., "An Experiment in Latency Reduction," *Proc. IEEE Infocom*, June. 1994, pp. 175-181.

[23] Watson, R.W., "The Delta-t Transport Protocol: Features and Experience," *Protocols for High Speed Networks*, Elsevier, 1989, pp. 3-17.

[24] Woodside, C.M., Ravinadran, K., and Franks, R.G., "The Protocol Bypass Concept for High Speed OSI Data Transfer," In participant's proceedings, IFIP Workshop on Protocols for High Speed Networks, Nov. 1990.

[25] Zitterbart, M., "High-Speed Protocol Implementations Based on a Multiprocessor Architecture," *Protocols for High Speed Networks*, Rudin, H. and Williamson, R. Eds., Elsevier, 1989, pp. 151-163.