A Protocol Structure for High-Speed Communication over Broadband ISDN

Zygmunt Haas

The *transmission speed in communication networks* has increased over the past decade from kilobits per second lines to hundreds of megabits per second; i.e., five orders of magnitude.[1] The processing speed of commercial Central Processing Units (CPUs) that can be employed as communications processors has changed only two to three orders of magnitude. This discrepancy in speed translates to "bottlenecks" in the communications process, because the software that supports some of the high-level functionality of the communication process is now several orders of magnitude slower than the transmission media. In other words, when the lines operated at 9.6 kb/s, the multilayer conventional protocols implemented in software were fast enough to match the speed of the lines. However, when the lines operate at hundreds of megabits per second, the mismatch in speed is so large that the advantage of high-speed lines is buried in the large processing overhead of high-level protocols, leading to long delay and low throughput. This change has shifted the bottleneck from the transmission to the software processing at the endpoints.[2]

Another trend that has influenced the design of communication networks is the potential of hardware implementation, i.e., Very Large Scale Integration (VLSI). It is much easier and cheaper today to implement large and fast communications hardware on a single silicon chip. However, this opportunity cannot be fully exploited with current protocols, which were developed for software implementation. The purpose of this article is to investigate the various possibilities of improving the performance of communications protocols and interfaces so that the slow-software-fast-transmission bottleneck can be alleviated, and to propose a new protocol architecture that is suitable for future high-performance communication.

The general trend of computing environments to move toward distributed and parallel processing systems is, and will be, largely responsible for the demand for increased performance. For example, a parallel processing system implemented on the fine-grain level requires delays on the order of microseconds. In a distributed processing system, large files may be required to be transferred between machines with very low latency. Thus, for the parallel and distributed processing environment, both the very low delay and large throughput are crucial. (Another bandwidth-demanding application that can be identified today is video. Video applications will become more and more important with traffic integration in packet-switched networks.)

Unfortunately, the impact of the overhead introduced by the Operating Systems (OSs) on the communication process strongly affects the application-to-application communication performance. The major sources of this overhead are [1–4]:

- Scheduling
- Multiple data transfer from/to the user[3]
- Overhead of entities management—timers, buffers, and connection states
- Overhead associated with division of the protocol processing into processes, including interprocess communication
- Interrupts
- Context switching

The reason for large OS overhead is the structure of the communication process in general and the implementation of network interfaces in particular. In other words, the CPU performs such an important role in the communication process that, as a consequence, there are too many interrupts, too many context switches, and too large a scheduling overhead. Network interfaces were invented to offload the CPU from the communication process. Unfortunately, they do only a partial job; interfaces are still being built that interrupt the processor for each received packet, leading to multiple context switches and scheduler invocations. (Another solution is to structure the process in a different way: to eliminate the scheduler calls by the interrupts and resolve the scheduling of the process that completed the communication at the "regular" scheduler invocations.) Some new proposals for interface architecture [2] considerably reduce the overhead associated with the OS. These proposals structure the communication process in much the same way that Direct Memory Access (DMA) is implemented; the CPU initiates a communication event, but has little to do in the actual information exchange process. The considerable reduction in the OS overhead that results from the new structuring of the communication process will probably have a crucial impact on the feasibility of providing multi-megabit-per-second bandwidth directly to the user (see "The Structure of the Network Interface" below for our ideas on a new interface structure).

The communication goal in the parallel and distributed system environment is to provide communication in which

[1]The ideas presented in this article may apply to end-to-end communication speeds far beyond Broadband Integrated Services Digital Network (BISDN) or even what is achievable today. In this vein, the article was presented at INFOCOM '90.

[2]We note that some of the processing overhead can be reduced by increasing the packet data unit size, so that the processing overhead is amortized over more data. However, this is not a universal solution, mainly because some of the overhead is byte-rather than packet-based. Moreover, for interactive traffic, only a small percentage of packets are of maximal length.

[3]It has been already emphasized several times in the literature (for example, [1] [2] that the number of data copies between buffers (i.e., the per octet overhead) has a crucial effect on the performance of a protocol.

0890-8044/91/0001-0064 $01.00 © 1991 IEEE

throughput is restricted only by the source capacity of the transmitter or the sink ability of the receiver. Also, the communication delay should be minimized. As stated, OSs are today the bottleneck of the communication process. However, once the OS bottlenecks are resolved, the performance required from communication systems will be so high that a new approach will be needed to the architecture of communication protocols and to network interface design to support high performance (high throughput and low delay). This argument answers the basic question of whether the current protocols are adequate for future high-speed high-speed networks. Local improvements in the protocol design might be adequate for current OS-limited systems; this will not be the case for future systems requiring throughput of hundreds of megabits per second directly to the user.

Along these lines, we propose an architecture that is an alternative to the existing layered architectures. The novel feature of the proposed architecture is the reduction in the vertical layering; services that correspond to the definitions of layers 4 to 6 in the International Organization for Standardization/ Open Systems Interconnection Reference Model (ISO/OSI RM) are combined into a single layer that is horizontally structured.[4] This approach lends itself more naturally to parallel implementation. Moreover, the delay of a set of processes implemented in parallel is determined by the delay of the longest process, not by the sum of all the process delays, as is the case in a sequential implementation. In the same way, the total throughput need not be limited by the lowest-capacity process, but can be increased by concurrently performing the function on several devices. Thus, a protocol structure that lends itself to parallel implementation has the potential to provide high performance matched to the requirements of the new generation of improved OSs.

The Challenge

The challenge is to propose a single[5] higher-layer[6] protocol that successfully provides communication over diverse networks: data rates ranging from kilobits per second to gigabits per second and network diameters from Local Areas Networks (LANs) to Wide Area Networks (WANSs). Also, the diverse requirements of many different applications need to be supported: connection-oriented and connectionless service, stream-like traffic and bursty traffic, reliable transport and best-effort delivery (error control and flow control), different data sizes, different delay and throughput requirements, etc. The required throughput is on the order of hundreds of megabits per second application-to-application (with a tendency toward gigabits per second, and the delay is on the order of hundreds of microseconds.[7] The intent is to propose a protocol structure that can support a very wide range of applications[8] and traffic types. Thus, some selectivity means in the protocol design is essential.

Let us discuss the above a little bit further. Assuming a very reliable subnetwork, the error recovery procedures can be simplified so that when there are no errors, very little overhead is incurred. However, this low overhead comes at the expense of having a large penalty in the event of an error. On the other hand, in networks with a high Bit Error Rate (BER) the recovery procedure for both the error and no-error cases should be minimized. This is what "success-oriented"[9] protocols mean: to minimize the overhead for successful delivery cases at the expense of a larger penalty for unsuccessful delivery attempts.

The reason for insisting on optimizing the performance[10] over such an extensive range of data rates (kilobits per second to gigabits per second) is that in the future we expect to have an enormous variety of networks. In other words, one cannot expect that, with the introduction of multi-megabit-per-second networks, Ethernets will (at least immediately) disappear. Even today, the range of communication is very large; 300 b/s modems are still being used.

Thus, in order to optimize the performance of the protocol for all diverse networks and applications, the protocol must consist of a set of versatile protocols that can be readily switched between. This is what we refer to in this work as "selective functionality protocols."

The work is organized in the following way. The next section outlines possible approaches to improving protocol performance. The section after that presents our horizontal protocol approach, while the selective functionality feature is described in the following section. A new approach to network interfaces is discussed briefly; then, a basic design example of the horizontally structured architecture is shown. The last section concludes the work.

The Possible Approaches

There are several possible solutions to overcome the slow-software-fast-transmission problem:

- One: Improve the performance of current high-layer protocol implementations [5] [6]
- Two: Design new high-layer protocols based on the current philosophy; i.e., software-based, layered structure [3] [7] [8]
- Three: Hardware implementation of high-layer protocols [9]
- Four: Reduction of high-layer protocols overhead (i.e., "lite" protocols)[11]

[4]Reduction in the number of layers was previously proposed. As discussed below, structing of the protocol into horizontally, conditionally independent functions is the novel feature of the proposed architecture.

[5]The reason for insisting on a single protocol is more than aesthetic. Much of the processing overhead (context switching, for example) can be avoided if a single implementation is used. Moreover, this approach may reduce compatibility problems.

[6]Above the network layer.

[7]Of course, because of the propagation delay, such a low delay requirement has little advantage for a WAN, and is necessary only in a LAN/MAN environment. However, because of the requirement that the protocol design be independent of the actual subnetwork being used, the stringent performance requirements need to apply to any communication.

[8]We are tempted to say "all" applications. However, future applications may introduce new requirements that may not necessarily be supported by the current design.

[9]This term refers to protocols that exploit the characteristics of reliable networks. Such networks, typically composed of fiber links and digital switches, have a very low error rate, deliver packets in order, and rarely drop packets.

[10]Here, "performance" refers to throughput, delay, and transmission efficiency.

[11]We note that a "lite" (i.e., lightweight) protocol is created by adjusting the number of states and the amount of control information that is passed between the protocol states in such a way, on one hand, to maximize the protocol functionality (to reduce the overhead of the software-based application layer) and, on the other hand, to minimize the overhead caused by the low-level implementation of the protocol states (for example, number of chips for hardware implementation or number of page faults for software implementation).

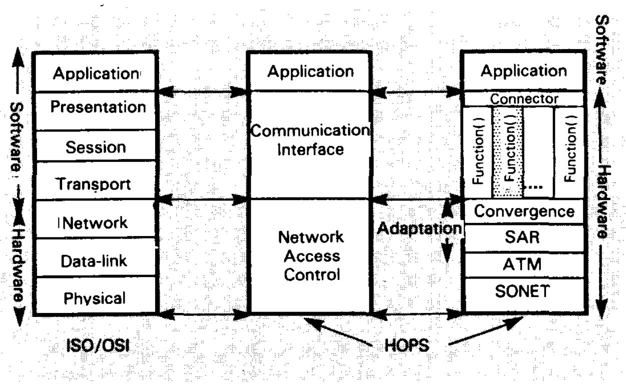

Fig. 1. HOPS.

- Five: New design philosophy that structures the high-layer protocols differently [4]
- Six: Migration of high-layer functionality to lower layers, in particular to the physical layer, when the functionality can be performed with lower overhead by the lower layer (for example, networks that perform some high-layer functions by trading the physical bandwidth, such as end-to-end flow control done at the physical layer in Blazenet [10])

The approaches were arranged according to how much they diverge from the conventional protocol design philosophy. Of course, there can be any combination of the above six approaches.

Note that there exists yet another approach: reduction of lower-layer functionality. (For example, [11] proposes to solve the link-by-link flow control problem by dropping excessive packets and correcting the packet loss at the higher level—transport—through retransmissions. We consider this approach, which is the opposite of the sixth approach, to be in general unable to solve the high-speed communication problem, since pushing the problems to higher layers introduces, in fact, larger delays (the implementation of higher layers is typically software-based, and thus slower). Also, in this specific case of the flow control example, the long timeout associated with detecting the dropped packets (which is done on an end-to-end basis) increases the overall delay. (We note, however, that pushing some of the functionality to higher layers simplifies the network design, since at the lower layers there is more traffic aggregation, and the total number of packets per second is larger, leading to more complex processing within the network. Thus, in some cases this approach may be appropriate.)

Every one of the six approaches outlined can potentially improve the protocol performance. However, major improvement in performance can be obtained by combining a few of the above approaches. For instance, even though changes in the current version of Transmission Control Protocol/Internet Protocol (TCP/IP) that reduce overhead might increase the protocol throughput to several hundred megabits per second, hardware implementation of the improved version will carry the improvement even further. Consequently, we believe that the correct answer to the question of how to implement high-speed protocols is an intelligent integration of several approaches.

In the next section, we consider a combination of the third, fouth, fifth, and (in a limited sense) the sixth approaches. The proposed architecture is based on the layers: the Network Access Control (NAC) layer, the Communication Interface (CI) layer, and the Application (A) layer. The NAC layer consists of all the services defined in and below the network layer of the ISO/OSI RM. The CI layer consists of the services defined by the transport, session, and presentation layers. The A layer corresponds to the conventional application layer. (The reason for the proposed three-layer structure is that the services of the NAC layer are mostly hardware-based, while the services of the A layer are software-implemented. Thus, the CI layer represents the boundary between the software and hardware. Therefore, in our view the model of the communication protocol architecture can consist of the three layers, where the NAC is hardware-based, the A is software, and the CI is a mixture of software and hardware. It is our belief that to achieve high-speed communication, the structure of CI must lend itself easily (i.e., little overhead) toward parallel hardware implementation and, as shown in the next section, the proposed three-layer structure provides a basis for such a parallel implementation.

Horizontally Oriented Protocol for High-Speed Communication

We propose here an alternative structure to the existing communication architectures. The central observation is that protocols based on extensively layered architecture posses an inherent disadvantage for high-speed communication. While the layering is beneficial for educational purposes, strict adherence to layering in implementation decreases the throughput and increases the communication delay. There are several reasons for this reduction in performance, among them (see also [4]) the replication of functions in different layers, performance of unnecessary functions, overhead of control messages, and inability to parallelize protocol processing.

The architecture presented here employs an approach quite different than that used in the extensively layered models; i.e., the proposed architecture has a horizontal structure, as opposed to the vertical structure of multilayered architectures. We refer to our architecture as Horizontally Oriented Protocol Structure (HOPS).

The main idea behind HOPS is the division of the protocol into functions instead of layers. The functions, in general, are mutually independent in the sense that the execution of one function can be performed without knowing the results of the execution of another. (Thus, intercommunication between the functions is substantially reduced.) For example, flow control and decryption are independent functions. If the dependence between two or more functions is such that the execution of one depends on the result of another, the function can still be conditionally executed. For example, packet resequencing is to be executed only if error control detects no errors. Thus, resequencing can be conditionally executed in parallel with error control, and at the end a (binary) decision is made on whether to accept or ignore the resequencing results.

Because of the independence between the functions, they can be executed in parallel, thus reducing the latency of the protocol and improving throughput.

Figure 1 shows the structure of HOPS. In this figure, the correspondence in services between HOPS and ISO/OSI RM is also shown. Thus, the CI of HOPS implements in hardware the services defined by layers 4 to 6. Further, in Figure 1 the detailed structure of the CI is shown. The figure also shows a possible ATM-based NAC implementation. The meaning of hardware implementation is not necessarily that HOPS is fully cast into silicon, but that specific hardware exists to perform the functions rather than relying on the host software. Thus, HOPS can be implemented as a collection of custom-designed hardware and general-purpose processors.

CI receives the raw information (unprocessed packets) from the NAC layer, which can be ATM-based. The central layer in HOPS is the CI. CI is divided into parallel, independent or

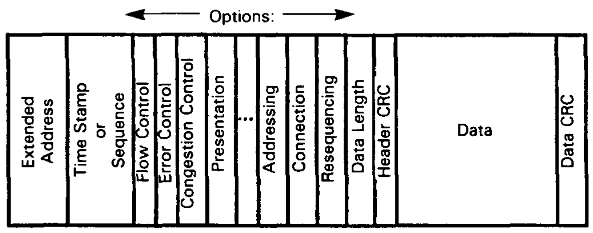

Fig. 2. HOPS packet format.

conditionally independent, functions. Before the results of the functions are passed to the Application layer, they are evaluated in the Connector. The Connector executes the conditional dependency among the functions, and passes the processed information to the Application layer.

HOPS is expected to lead to high-performance implementations for several reasons. First, because of the horizontal structure of functions, the delay of a packet processing is determined by the slowest function rather than the sum of all delays. This is achieved in HOPS by the independence of functions. Thus, a function need not wait for a result of another function before its execution can begin. Second, the throughput can easily be improved by increasing the number of units of capacity-limited functions. Such an increase is rather natural in parallel-structured CI. Third, because of the compression of layers, much of the replication and overhead are eliminated (e.g., buffering on different layers). Fourth, the horizontal structure lends itself to parallel implementation on separate, possibly customized, hardware. The parallel implementation by itself has the potential of lowering the processing delay and increasing the processing throughput. Also, the overhead associated with switching between the execution of functions is grossly eliminated, as are the communication messages between the processes. Finally, the selective functionality feature (discussed in the next section) can eliminate unnecessary function processing.

We also believe that in order to achieve the limit of performance, one should implement the HOPS-structured protocols in custom-designed hardware [9].

It should be noted that HOPS, as well as other solutions based on the fourth, fifth, and sixth approaches described earlier, are not compatible with the ISO/OSI model and, in fact, violate the model's boundaries.

HOPS as a Selective Functionality Protocol

Because HOPS is intended to support communication over diverse networks and for diverse applications, a single protocol cannot provide optimum performance. For example, retransmission policy depends on the quality of the network: selective retransmission is better for networks with large average BER, while go-back-*n* may be beneficial in a very reliable environment. Moreover, the requirements for a protocol may change with time and space; for example, increasing congestion may change retransmission policy, or the required error control mechanism may differ from subnetwork to subnetwork. Consequently, what we propose is a "protocol with a menu," whereby a user will request some combination of functions needed to achieve some particular level of performance. For instance, the function "retransmission" may be designed to receive the following values: selective, go-back-n, go-and-wait, none, any, etc. The Network Interface (NI) has to decide on the particular retransmission policy required. If the NI has some knowledge about the subnetworks the packet is going to travel on, then the NI can make an intelligent decision on the required policy. The NI can change its decision with time, if it learns that the conditions have changed or that its previous decision was incorrect.

The function values are communicated throughout the network by means of the special option field in the packet format, as shown in Figure 2. Note that the packet format shown in Figure 2 is the only packet format required by HOPS for the whole CI layer; i.e., the packetization overhead present in multilayered protocols is severly reduced in our approach. The values are decoded separately, in parallel, and "on the fly" at the packet arrival time.

There is another very important advantage of the selective functionality feature: possible compatibility with current protocols. It cannot be expected that there will be an immediate transfer from the rooted architectures and protocols. Thus, protocols like Transport Protocol No. 4/Connectionless Network Protocol (TP4/CLNP) will continue to exist, and it is of paramount importance that the current and new high-speed-oriented protocols interwork. The selective functionality approach enables one to compose any protocol from the extended menu. Thus, for example, communication between a TP4/CLNP site and a HOPS site can easily be achieved by an adaptation layer (to be abolished with time) providing simple translation of the TP4/CLNP packets to the HOPS format. Since by virtue of the selective functionality feature, HOPS is a superset of all the TP4/CLNP functionalities, such a translation is possible.

Initial considerations of the HOPS architecture suggest that delay and throughput will be determined by the slowest capacity-limited functions. However, as opposed to conventional protocols, these limits are not fixed, and they change with the particular functionalities required by the network/application pair. Moreover, since networks are dynamic entities, the required correction mechanisms may be adaptively adjusted by the end-to-end HOPS functionalities. Thus, transmission over a highly reliable network may skip most of the overhead associated with resequencing. Furthermore, custom-designed implementation may increase the performance of the limiting functions.

The Structure of the Network Interface

The NI is the hardware that interconnects the network and the host by receiving the packets addressed to the host and transmitting the host output traffic over the network (see Fig-

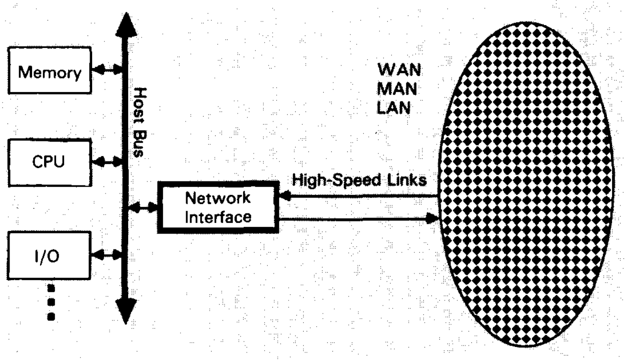

Fig. 3. Network interface.

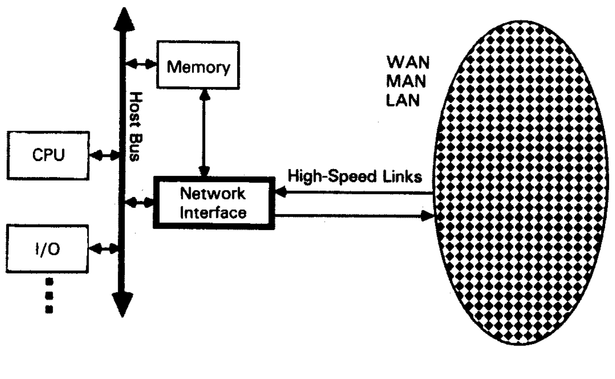

Fig. 4. Proposed communication model.

ure 3). The NI proposed in this work also performs the high-layer protocols. Thus, the NI is the actual hardware that implements the CI of HOPS.

The goal of the NI is to offload the protocol processing from the CPU. For example, in the Network Adapter Board (NAB) [2], the CPU initiates information exchange but does not participate in the actual transfer of data. The NAB does substantially reduce the host load, thus increasing the interface bandwidth. We want to push this idea even further, and eliminate even more of the CPU's involvement in the communication process. In other words, we intend to:

- Eliminate CPU involvement in processing the transport functions
- Eliminate CPU involvement in processing the presentation functions
- Reduce CPU involvement in processing the session functions
- Reduce the overhead associated with the communication between the NI and the OS (e.g., interrupts and scheduler)
- Eliminate the buffering and copying of information from OS space into user space.

The elimination of CPU involvement in processing the transport and presentation functions is done by fully performing these functions within the CI layer by the NI, as explained earlier.

The reduction of CPU involvement in processing the session functions is a bit tricky, since obviously some processing must be done by the OS. For example, opening a new session usually involves space allocation to the user. This is done by the OS. However, most of the bookkeeping and some processing can be done by the NI; for example, negotiation of parameter values and management of the dialogue once the connection is established.

The reduction in communication between the NI and OS is performed in several ways. First, the NI keeps a pointer to the user space, so that it can write the received data directly to the user buffers. This eliminates interrupts when data is received. (This also eliminates the overhead associated with copying the data from OS space into user space.) Second, the NI has access to the scheduler and monitor tables, directly adjusting the table upon reception of data. When the scheduler is invoked, it checks on the status of the pending communication processes by referring to the NI-accessible table. Finally, the scheduler itself can be implemented as a different module.

Some remarks are called for here. First, it may be necessary to keep the user buffers aligned on the word boundary. Second, either paging is inhibited (it may make sense when the data to be communicated is small, or when the machine is dedicated to a single job) or the NI must be informed of paging (so that it does not write into space that does not belong to the destination user). If paging occurs, the OS needs to provide a buffer assigned to the user. (This might be a disadvantage, since once the user is running again, the data need to be transferred between the OS and user spaces.) Another approach is to use communication buffers located in common space that belongs to the OS (more specifically to the NI) and the user process. The NI writes the received data into these buffers, which can be directly (i.e., without copying) accessed by the user. Third, there must be a very intimate relationship between the OS and the NI; the NI must have access to OS tables. The NI access to the OS data structures may be system-dependent. Finally, the resequencing, if needed, can be done directly in the user buffers. The NI leaves "holes" in the buffer for unarrived packets, and indicates arrival of the last in-sequence data.

The communication model associated with this work is based on the assumption that as little work as possible should be done by the CPU in the communication process. Thus, for example, when a process running on a workstation requires a file, the process initiates a request. The request is executed by the local NI, bringing the file into the process space with some diagnostic information in a special register. The NI serves as an agent on behalf of the process. (This is accomplished, as shown in Figure 4, by the direct connection between the NI and the memory). The NI is responsible for opening a connection (if necessary), performing the parameter negotiation, checking the validity of the incoming data, retransmitting (if necessary), performing special functions like encryption/decryption and format translation, and closing the connection (if necessary). The CPU is not involved in any stage of the communication process, and in fact, the CPU sees a file fetch as a simple read operation. (In this sense, there is some limited similarity to Memnet [12].)

Consequently, in the transmission process, the CPU passes data and control information to the NI, but has no hand in the actual execution of the information exchange. (The CPU can, however, receive a code indicating whether the other side actually received the data). On the receiving side, the processing depends on the type of data received. If the data is:

- Request for new connection: The NI sets the required information in the Table Of Flows (TOF),[12] and will interrupt the CPU only if some other action is required to be performed. For example, when a new session open request arrives, the request will generate a new entry in the TOF; however, the CPU will be interrupted to allocate space for the new session.
- New connection-oriented packet: The packet is placed directly in the process space, without CPU intervention. If necessary, control information is set in the NI for future process incarnation.
- New request-response:[13] This is managed directly by the NI by interpreting the request and directing it to the service. If the service is not recognized, the CPU might be interrupted.
- New datagram packet: This is the same as a request-response packet.

[12]The TOF is a data structure maintained in the NI to keep track of the existing connections.

[13]In the transactional communication model [7], a request sent to a server and the server response are both performed in the connectionless mode.

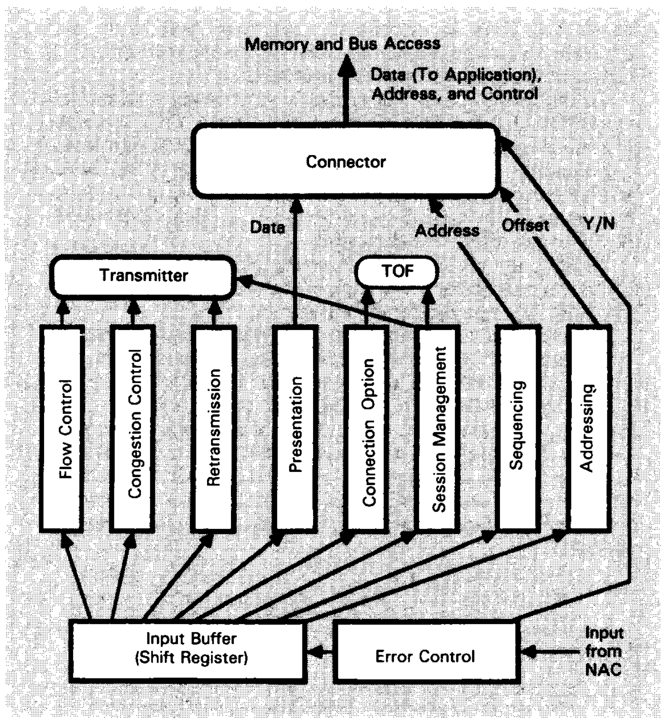

Fig. 5. General structure of HOPS.

Example of HOPS Design Implementation

In this section, we provide an example to illustrate how the HOPS architecture can be implemented. The implementation presented here includes the receiver side only. Also, it is somewhat simplified in the sense that not all the data structures are formally defined.

The following functions are incorporated into the design:

- Error control
- Retransmissions
- Connection option
- Sequencing
- Flow control
- Addressing
- Presentation
- Session management
- Congestion control

The structure of the design, presented in Figure 5, shows the flow of information between the various blocks that implement functions. The parallel structure is emphasized. In what follows, each function is described by its attributes: values that the function can receive, functionality performed, what input is received by the function, and what output is produced.

Error Control

Values: *error-control*, *detection*, *n-correction*, *m-detection*, and *n-correction.*[14]

[14]An *n-correction* algorithm is capable of correcting *n* errors in a packet and an *m-detection* and *n-correction* algorithm is capable of correcting *n* errors and detecting *m* errors.

Functionality:

- Checks the computed and packet Cyclic Redundancy Checks (CRCs).
- If *error-control* = = *detection*, then forward the data to the output only if CRCs match. Otherwise, reject the data.
- If *error-control*= = *n-correction*, then attempt data correction and forward the data to the output if successful.
- If *error-control* = = *m-detection* and *n-correction*, then if *number-of-errors* $\leq$ *n* attempt data correction, if *n* $<$ *number-of-errors* $\leq$ *m* reject the packet and notify application.

Input: Options (*error-control*), CRC, and data.

Output: Data (if successful operation).

Note: Error control is done "on the fly" before the packet is inserted into the input shift register (see Figure 5).

Retransmissions

Values: *selective*, *go-back-n*, *none*, and *any*.

Functionality:

- Keeps track of data flows
- If timer expires, issues retransmission request according to value

Input: Options (*retransmission-policy*).

Output: Control packets in the reverse direction with retransmit requests.

Connection-Option

Values: *datagram*, *connection*, and *request-response.*

Functionality:

- Keeps track of the data flows by recording each flow in the TOF

Input: Options (*connection-option*).

Output: TOF entries.

Sequencing

Values: *yes* and *no.*

Functionality:

- Provides sequencing in the user memory-based buffer

Input: Options (*sequencing*, *sequence-nr*, or *time-stamp*) and address.

Output: Address offset.

Flow Control

Values: *window*, *permits*, *on-off*, *none*, *window-size*, and *window-value*.

Functionality:

- Keeps track of user buffer size per flow, and performs the required flow control scheme
- Keeps track of window value for window-controlled flows

Input: Options (*flow-control*, *window-size*, *window-value*).

Output: Flow-control information to the transmitter.

Addressing

Values: *none.*

Functionality:

- Resolve the address/port/process

Input: Address.

Output: Memory address.

Note: It is assumed that multicasting is provided by the NAC layer.

Presentation

Values: various presentation standards and encryption algorithm.

Functionality:

- Data translation according to the presentation standard

Input: Options (*presentation* and *encryption algorithm*), data.

Output: Data (after presentation operation).

Session Management

Values: various session-related values (bandwidth, timers, etc).

Functionality:

- Bandwidth allocation
- Management of session parameters

Congestion Control

Functionality:

- Evaluate parameters for congestion control operation [5]

Conclusions and Summary

In this work, we have presented a rationale for the need to improve the performance of high-level protocols. In particular, changes in the design of network interfaces are expected to increase the feasible throughput available to the application to hundreds of megabits per second. Corresponding improvement in the architecture of operating systems promises to reduce and eliminate the current OS overheads.

The current structure of communication protocol suites, based on vertically layered architectures, posseses an inherent bottleneck due to the layering process. Functions are in some cases unnecessarily replicated in different layers, unnecessarily performed in some circumstances, and add an unnecessary interprocess communication burden to protocol execution. Moreover, the major disadvantage of the layering process is the relative difficulty in parallel implementation of the layered structure. Parallelism offers the potential of increased protocol processing rates, reduced processing latency, and reduced processing overhead.

We have presented an alternative approach to the existing architectural models, named Horizontally Oriented Protocol Structure. The architecture is based on three layers: Network Access Control, Communication Interface, and Application. The Communication Interface is the heart of the architecture, and is based on independent functions that can be performed in parallel. Thus, the architecture presented here lends itself naturally to parallel implementation. A design example of the Communication Interface was presented.

In the future, there will probably exist many distinct network types that will provide communication to different types of applications. In such a reality, it is beneficial to have a protocol versatile enough to easily and adaptively adjust itself to the kind of performance demanded by the network-application requirements. The selective functionality protocols introduced in this work offer such behavior.

Finally, the new network interface philosophy, which will offload even more of the burden of communication processing from the OS, was briefly discussed.

It is our belief and hope that the ideas presented here will contribute to the progress in developing communication networks, which resemble an extension of a computer bus, and components, which are capable of providing very high bandwidth (on the order of hundreds of megabits per second) directly to the user. Moreover, such networks can provide a vehicle for today's and tomorrow's implementations of ISDN. If this vision becomes reality, the communication process will cease to be the limiting factor in the latency of a process execution.

References

[1] S. Heatly and D. Stokesberry, "Analysis of Transport Measurements Over a Local Area Network," *IEEE Commun. Mag.*, June 1989.

[2] H. Kanakia and D. R. Cheriton, "The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors," *Proc. SIGCOMM '88*, Stanford, CA, Aug. 16–19, 1988.

[3] K. Sabnani, M. H. Nguyen, and C. D. Tsao, "High-Speed Network Protocols," *6th IEEE Int'l. Workshop on Microelectronics and Photonics in Commun.*, New Seabury, MA June 6–9, 1989.

[4] A. Tantawy, H. Meleis, M. El Zarki, and G. Rajendran, "Towards a High-Speed MAN Architecture," *ICC*, Boston, MA, June 11–14, 1989.

[5] V. Jacobson, "Congestion Avoidance and Control," *Proc. SIGCOMM '88*, Stanford, CA, Aug. 16–19, 1988.

[6] D. D. Clark, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *Proc. 13th Conf. on Local Comp. Networks*, Minneapolis, MN, Oct. 10–12, 1988.

[7] D. R. Cheriton and C. L. Williamson, "VMTP as the Transport Layer for High-Performance Distributed Systems," *IEEE Commun. Mag.*, vol. 27, no. 6, June 1989.

[8] D. D. Clark, M. L. Lambert, and L. Zhang, "NETBLT: A High Throughput Transport Protocol," *Proc. SIGCOMM '87* and *Commun. Rev.*, vol. 17, no. 5, 1987.

[9] G. Chesson, "XTP/PE Overview," *Proc. of 13th Conf. on Local Comp. Networks*, Minneapolis, MN, Oct. 1988.

[10] Z. Haas and D. R. Cheriton, "Blazenet: A Packet-Switched Wide-Area Network with Photonic Data Path," *IEEE Trans. on Commun.*, June 1990.

[11] M. El Zarki and N. F. Maxemchuk, "Routing and Flow Control in High-Speed Wide Area Networks," *Proc. IEEE*, Jan. 1990.

[12] G. S. Delp, A. S. Sethi, and D. J. Farber, "An Analysis of Memnet: An Experiment in High-Speed Shared-Memory Local Networking," *Proc. SIGCOMM '88*, Stanford, CA, Aug. 16–19, 1988.

Biography

Zygmunt Haas received his B.Sc. in electrical engineering from Technion in 1979 and M.Sc. in electrical engineering from Tel Aviv University in 1985, both summa cum laude. From 1979 till 1985, he worked for the government of Israel. In 1988, he received his Ph. D. from Stanford University, and subsequently joined AT&T Bell Laboratories in Holmdel, New Jersey, where he is now a Member of Technical Staff in the Network Systems Research Department. His interests include high-speed communication, high-speed protocols, lightwave networks, and traffic integration.