Architectural Considerations for a New Generation of Protocols

David D. Clark and David L. Tennenhouse
Laboratory for Computer Science, M. I. T.

Abstract

The current generation of protocol architectures, such as TCP/IP or the ISO suite, seem successful at meeting the demands of todays networks. However, a number of new requirements have been proposed for the networks of tomorrow, and some innovation in protocol structuring may be necessary. In this paper, we review some key requirements for tomorrow's networks, and propose some architectural principles to structure a new generation of protocols. In particular, this paper identifies two new design principles, Application Level Framing and Integrated Layer Processing. Additionally, it identifies the presentation layer as a key aspect of overall protocol performance.

1 Introduction

Historically, there has been a sharp distinction between the network architectures used in universal access environments, such as the telephone network, and the architectures developed within the computer communications community. We believe that both communities would benefit from a unified architecture that addresses new communication requirements. We have identified three key requirements that will stress the existing protocol architectures. Future networks will have considerably greater capacity, will be based on a wider selection of technologies, and will support a broader range of services.

The principle design goal for many research projects, and some commercial products, is gigabit operation, both in aggregate over trunks and to individual end points. As networks proceed to higher speeds, there is some concern that existing protocols will represent a bottleneck, and various alternatives have been proposed, such as outboard protocol processors [1, 11] and new protocol designs [2, 14, 12]. Since there is no clear consensus on the real source of protocol overhead, it is hard to validate these various solutions.

Another requirement is that the protocols of tomorrow operate over the range of coming network technology. An obvious example of emerging technology is Broadband ISDN [19, 6] which is based on Asynchronous Transfer Mode (ATM). This network technology provides a switching mode somewhat different from classic packet switching in that the data to be transferred is divided up into small (currently 48 byte) fixed size elements for transmission. A more futuristic network technology is wavelength division over fiber.

Finally, there is the requirement for service integration. In the future, a single end system will be expected to support applications that orchestrate a wide range of media (voice, video, graphics, text) and access patterns (interactive, bulk transfer, real-time rendering). This cross-product of media and access will generate new traffic patterns with performance considerations, such as delay and jitter tolerance, that are not addressed by present architectures.

In summary, future networks must exhibit a significant degree of flexibility and be based on an architecture that admits a wide range of application demands and implementation strategies. In the remainder of this paper, we identify some key principles for the elaboration of such an architecture.

2 Structuring Principles for Protocol Architectures

Most discussions of protocol design refer to the ISO reference model [9], or some other model that uses layering to decompose the protocol into functional modules. There is, however, a peril in using only a layered model to structure protocols into components, which is that layering may not be the most effective modularity for implementation.

It is important to distinguish between the architecture of a protocol suite and the engineering of a specific end system or relay node. The architecture specifies the decomposition into functional modules, the semantics of the individual modules and, the syntax used to effect the protocol. There should be no *a priori* requirement that the decomposition of the engineering design of a given system correspond to the architectural decomposition of the protocol. In the case of layered architectures, practical experience [4, 17] provides strong support for alternative engineering designs. Unfortunately, the structure of the protocol architecture may unnec-

*This research was supported by the Defense Advanced Research Projects Agency, monitored by the National Aeronautics and Space Administration under contract No. NAG2-582, and by the National Science Foundation under grant NCR-8814187.

essarily constrain or complicate the engineering alternatives available to the implementor [18].

We conclude that a key architectural principle should be flexible decomposition: the deferral of engineering decisions to the implementor and the avoidance of inessential constraints. Within the domain of a single protocol architecture, different engineering schemes may be appropriate depending on the functions being performed (relay *vs.* end systems), the performance requirements of the applications, and the resources available to the designer.

In the next section of this paper we examine the core functions of a protocol architecture and distinguish between essential and inessential constraints. The objective throughout is to identify architectural approaches that might lead to streamlined engineering designs that are customized to suit their environments. Therefore much of this paper discusses engineering issues, with particular emphasis on the design of end systems.

3 Protocol Functions

The core function of protocols is to transfer application information among machines. Thus, an obvious way to review the function of protocols is to focus on the data transfer phase of protocol operation and to observe and catalog the operations that occur there. This leaves for separate consideration those operations such as session initiation, service location, and so on, which, while very important, do not occur at the same time as data transfer.

A distinction which seems to have great relevance is the separation of data transfer functions into two groups: those which actually read or modify the data, which we will call **data manipulation** and those functions that regulate the transfer, which we will call **transfer control**. Despite the great complexity of many protocol suites, there are in fact not that many manipulation and transfer control functions that actually occur at the time of data transfer. Based on existing practice, the most important and universal of these functions are the following.

Data Manipulation

There are six manipulation functions which are generally found in protocols. Note that while we list them as separate, several of them may be accomplished in the same operation.

Moving to/from the net — the most obvious and unavoidable manipulation function is the actual transfer of the data in or out of the network itself, which usually involves some sort of serial-to-parallel transformation. This function is usually performed in custom hardware which decouples the timing of the network from that of the host.

Error detection — data is usually protected by some sort of checksum or related function, which is computed by reading the data. When an error is detected, there may be an error correction stage that also involves a data manipulation.

Buffering for retransmission — at the sending end of a transfer, many protocols reserve a copy of the data, so that if it is lost in transit it can be resent.

Encryption — either for privacy or integrity, data is encrypted. This function, if implemented, can sometimes also provide error detection.

Moving to/from application address space — most commonly, data is not moved directly between user address space and network interface, but is moved to some intermediate buffer in system address space. This may be a fundamental requirement on input, as discussed below, but is often dictated by the details of the system I/O structure.

Presentation formatting — most data transfers require that the data be reformatted into some common or external data representation. In the standards world, obvious examples include the ISO ASN.1 [10], or the SUN XDR [16]. Some data is sent across the network in what is called "image," "internal," or "raw" mode, without such a conversion, but even a universal standard such as ASCII may require reformatting.[1]

These six steps are generally associated with different layers in the protocol suite, but they represent a common class of overhead, since they all involve reading and writing the data, and in some cases moving it from one part of memory to another. Presentation formatting, for example, usually involves moving the data into a new area, since the result of formatting may, in general, be of a different size than the original.

Transfer Control

There are a number of operations that are directly related to the detailed control of the data transfer phase. We will focus on these "transfer control" operations and set aside the remaining control functions, such as connection initiation. The most common transfer controls are the following:

Flow/Congestion control — To protect both the network and the receiver, the sender must be regulated to send no faster than the data can be accommodated. The minimal in-band control function involves the pacing of the data at the transmitter and the monitoring of arrivals at the receiver. The actual computation and negotiation of the transfer rate can be performed on an out-of-band basis.

Detecting network transmission problems — Networks, especially packet switched networks, have specific failure modes. Data may be lost due to congestion overflow, and it may be reordered or duplicated as a part of processing. It is sometimes convenient to think of lost packets as a different problem from reordered packets, but we will have more to say on this later.

Acknowledgement — A common control function is positive acknowledgement of data receipt. This control is sometimes thought of as universal. In truth, it is but one of many methods for dealing with network errors. However, it is a step with considerable complexity and thus deserves separate mention.

[1] Since ASCII is vague on the representation of its newline convention, the Internet protocols [13] require a conversion from internal ASCII to external ASCII.

Multiplexing — Several hosts share a network, and several processes share one host. Thus several data streams may interleave entering or leaving a host. These must be delivered properly, both to insure basic function, and to prevent security problems arising from mis-delivery.

Timestamping — Some real-time protocols rely on packet timestamps to support the regeneration of inter-packet timing.

Framing — Encapsulation-based protocols require that frame boundaries be conveyed between sending and receiving entities.

Some aspects of transfer control are required as a part of data transfer, but need not be performed in lock-step with the actual transfer operations. For example, the computation of flow control parameters is obviously a part of data transfer, but it need not be synchronized with the manipulation of individual data units. We will use the term "in-band" for those controls that are integral to manipulation steps, and the term "out-of-band" for those controls that can be somewhat decoupled from the transfer and done in the background. One of our design goals is to reduce to a minimum the number of in-band control operations, thereby increasing the number of engineering options for the manipulation steps.

4 Performance – Control *vs.* Manipulation

The various control operations may interact in complex ways, but in fact the list is very short. It may come as a surprise, given the apparent complexity of protocol specifications, but examination of efficient implementations, for example, the current implementation of TCP for Berkeley BSD Unix [3], show that not many instructions are required for the in-band control operations.

Consider the following brief review of the control steps for an incoming packet. First, the packet must be properly demultiplexed or dispatched. This requires that one or more fields in the packet be examined, and a local state structure retrieved. A manipulation step is required to check for data errors, but the normal case (no errors) does not require control action. The only control test is to verify that the packet is in order, which is a simple comparison with the local state. The next step is to compute the proper acknowledgement information, and trigger the sending (now or later) of an acknowledgement packet. This step may also involve the computation of some flow control information.

None of these control steps is computationally complex. The largest, based on experience, is the computation of flow control parameters, which most commonly occurs at the sender of the data. But the total path lengths are tens, not hundreds of instructions.[2]

In contrast, the data manipulations are much more processing intensive, since they involve touching all the bytes in the packet, perhaps several times. A typical large packet today might have 4000 bytes, or 1000 long words of data. Thus to touch the data even once is 1000 memory cycles,

[2]The true costs may be somewhat higher as a result of indirect overheads associated with context switching, the invalidation of translation look-aside entries, cache depletion, etc.

	μVax	R2000
Copy	42	130
Checksum	60	115

Table 1: Speed in Mb/s for manipulation operations.

to copy is 2000, and there may be several such steps. One is thus led to the conclusion that the manipulation steps are the more obvious target for overhead reduction and that control need be examined only to the extent that it makes the burden of manipulation worse than necessary. This conclusion is controversial, but is supported, for example, by the work reported in [3].

Some specific numbers make the actual cost of data manipulation more concrete. In TCP, two fundamental manipulation operations are word-aligned copies and checksums, the latter requiring a read of the data, with a very simple computation. Table 1 reports some actual measurements of the speeds of these operations, in megabits per second (the normal rating for protocols, if not hosts) using hand-coded unrolled loops, for two machines, a μVax III and a MIPS R2000.

The copy cost provides almost an absolute upper limit on the throughput that can possibly be achieved for any CPU. It is difficult to see how a useful protocol can be devised that does not need to move the data at least once.

If the data manipulations listed above are examined, one conclusion is that they are usually thought of as distinct operations, since they occur in different protocol layers. But this is not necessarily the most efficient design approach. With current RISC chips, which pay a very high cost to read or write memory, it is more efficient to read the data once and perform as many manipulations as possible while holding the data in cache or registers. This approach may prove even more effective with the advent of super-scaler processors that perform a number of operations during each memory cycle.

For example, in the case of the MIPS processor above, the copy and the checksum ran at 130 and 115 Mb/s, respectively. If they were done separately, the result would be an effective throughput of about 60 Mb/s. In comparison, a hand coded unrolled loop that did both operations at once ran at 90 Mb/s. This may not seem a dramatic improvement, but the effect would be much more beneficial if several of the necessary manipulation steps were combined. [3]

Presentation Conversion

One manipulation step has a key impact on performance — presentation conversion. This is because it is often so very costly. Again, some simple experimental numbers may make clear the typical costs. Using the R2000 as an experimental CPU, we compared the cost of a word-aligned copy (the

[3]The actual savings may be somewhat higher. Our simple analysis does not take into account the previously noted indirect overheads associated with the separate implementation of protocol operations.

basic manipulation), with various sorts of presentation conversions. As noted above, the word-aligned copy ran at 130 Mb/s. In comparison, a hand coded conversion routine to translate an array of integers into ASN.1 ran at 28 Mb/s, a factor of 4-5 slower. Similar results for other processors are reported in [8].

Another experiment was to examine the performance of a protocol stack comprising the current Unix TCP package and the ISODE implementation of the OSI upper layers. A comparison of throughput with and without significant presentation conversion[4] showed that about 97% of the total protocol stack overhead was attributable to the presentation conversion function. In effect, the conversion-intensive case ran about 30 times slower. In summary, the two experiments provide a range of relative performance, presentation *vs.* lower layers, within which one might expect real systems to perform.[5]

The argument about combining manipulation operations takes on a different nature when these large presentation costs are included. Returning to the R2000 integer conversion example, the basic ASN.1 conversion routine ran at an average rate of 28 Mb/s. Adding the TCP checksum manipulation to the code, so that it converted and checksummed in one step, only slowed the result to about 24 Mb/s.[6]

5 Presentation Processing

Because of the large costs that we currently see for presentation conversion, it becomes clear that to achieve good performance, one of two things must happen. Either the presentation layer must be omitted, or it must be the focus of performance tuning. In fact, most applications that attempt to achieve high performance today take the first approach and transfer data in some raw or image format. However, there is a loss of generality in eliminating the presentation layer from all high performance protocols. We thus consider how to maximize the speed of presentation conversion, before looking at the other option. One important activity is the optimization of presentation conversion, through the tuned implementation of existing standards or the introduction of alternatives, such as the light weight transfer syntax described in [8].

A key aspect of presentation conversion is that it needs to be done in the context of the application. Part of presentation conversion is moving the data to or from the application data space. In some cases, such as file transfer, this is a simple operation of placing the converted data in sequential locations. In general, however, the presentation is to or from various language-level variables. For example, in the case of Remote Procedure Call support, the transferred data represents the arguments and results of a procedure call, and must be moved to the stack of the application process. In some cases, only the application will know what the sequence of data items is, so that the actual sequence of presentation conversions must be driven by application knowledge.

Mechanically, what this means is that, if a presentation conversion is involved, the application process (rather then the system kernel or an outboard processor) will be the usual bottleneck in overall network throughput.[7] This point should not impact the ability to attack performance using special coding techniques such as hand-coded unrolled loops, but it can have a great impact when one takes into account the pipelined nature of the data manipulation steps. On the receiving end, if the application cannot run whenever data arrives from the network, it will fall behind, and since it is the bottleneck, it will never catch up. A design goal must be, therefore, to design protocols so that the application is not prevented from performing presentation conversion as the data arrives.

The Importance of Lost and Mis-Ordered Data

The relation between reordering and presentation processing is in fact a key architectural consideration in protocol design, a consideration which has not received explicit attention in the past. In most current protocol designs it is difficult to keep the presentation conversion running in real-time, since data loss or reordering prevents immediate processing as the data arrives. For this reason, we will consider in more detail some aspects of loss and reordering.

Why does data get out of order in the first place? There are two answers. Data may be mildly out of order because two of the data units, e.g. packets, are reordered in some switching node. The more drastic reordering occurs as an indirect result of a lost packet. If a packet is lost, any data after it can be thought of as being "out of order" with respect to the lost packet.[8] Present day applications are not equipped to deal with packet loss or reordering. Instead, lower layer protocols such as TCP hold up all the following data, request retransmission of the lost packet, and then proceed with final manipulation after it arrives. Certainly, with TCP, any presentation conversion can only occur after TCP has completely reordered and recovered the incoming data. Thus, a lost packet stops the application from performing presentation conversion, and to the extent it is the bottleneck, it can never catch up.

To permit some manipulations to be performed in the presence of mis-ordering or loss, the typical approach is to put "synchronization points" in the data stream, points where manipulation can start even if data before that point is missing. Thus for example, error detection checksums are usually computed separately for each packet, permitting each packet to be checked in isolation. The same is true for many encryption schemes, though some sort of chaining is often used to guard against malicious reordering. These are examples from the lower layers of protocols; in brief our goal is to extend this idea up through presentation to the application.

[4] The baseline case transferred a very long OCTET STRING while the conversion-intensive case transferred an equivalent length array of 32 bit integers.

[5] To be fair, the present ISODE package is a prototype toolkit. If it were subjected to the degree of tuning that has been applied to the TCP package, the gap between its relative performance (30:1) and that of the hand-tuned code (4-5:1) might be significantly reduced.

[6] Here again, the indirect overheads associated with separate implementation may be an important consideration.

[7] This point is overlooked when claims for performance are made based on memory-to-memory transfers at the transport level.

[8] In the remainder of this paper our use of the term "out of order" includes this case where no reordering has occurred but an intervening data unit has been lost in transmission.

What should the unit of synchronization be to permit the application to perform presentation conversion on out of order elements? One could consider some scheme based on the transmission unit, or packet, for this purpose, but this does not seem fruitful. Where the packet was a basic unit of data transmission (and data reordering), it was natural to consider the packet as the unit of manipulation as well. This principle will not hold in the future. "Classic" packet switching is not the only method of multiplexing that will be used. Asynchronous Transfer Mode, or ATM, segments data into small units called cells, with a data payload of 48 bytes.[9] This is probably too small a unit of data to permit manipulation operations to be synchronized on each cell. Fiber multiplexing based on wavelength division need not provide transmission framing at all.

We adopt another design approach. Instead of starting with the transmission unit such as the packet, we propose to start with the application (which is, after all, the reason for the data transfer in the first place) and consider what its basic requirements are for dealing with lost or re-ordered data.

Application Processing of Mis-ordered or Incomplete Data

The manner of coping with data loss is highly dependent on the needs of the application. The classic transport model is that the protocol will suspend delivery of data to the receiving client, and retransmit from a copy of the data saved at the sender. But this is not the only pattern. Another option is for the application to accept less than perfect delivery and continue unchecked. This will work for real-time delivery of video and voice. Another option is for the sending end to retransmit, but for the application rather than the transport protocol to provide the data. This permits the sending application to recompute the lost data values, rather than buffering them. Finally, in some real time situations, the application may not literally retransmit lost data, but it might send some new data which eventually "fixes" the consequences of the original loss.

A general purpose data transfer protocol ought to permit any of these options to be selected: buffering by the sender transport, recomputation by the sending application, or proceeding without retransmission.

What can be concluded from this goal? If the application is to have the option of dealing with a lost data unit, either by reconstituting it or by ignoring the error, then losses must be expressed in terms meaningful to the application. Transport protocols such as TCP do not have this feature. TCP numbers the bytes in the data stream, and uses these numbers to achieve data ordering and retransmission. However, these numbers have no meaning to the application. This is true because the presentation layer, which lies between the application and the transport, converts the format of the data in a way that, in general, changes the size of the elements. So without understanding the transformation that the presentation layer has done, it is impossible for the application layer to relate lost data at the transport to the equivalent application data.

[9] We do not believe that ATM cell loss rates will be so low as to eliminate order maintenance as a protocol issue. Although the draft CCITT recommendations proscribe cell reordering, they make significant provisions for cell loss detection, primarily within the "Adaptation Layer." The net cell payload, after adaptation, is 44-46 bytes.

Application Level Framing

The way to avoid this problem is for the lower layers such as presentation and transport to deal with data in units that the application specifies. In other words, the application should break the data into suitable aggregates, and the lower levels should preserve these frame boundaries as they process the data. This proposal will call these aggregates Application Data Units, or ADUs. ADUs will then take the place of the packet as the unit of manipulation. We call this design principle Application Level Framing.

What defines a suitable size for an ADU? The fundamental characteristic of our definition is that each ADU can be processed out of order with respect to other ADUs. This rule permits the ADU boundaries to take the place of the packet boundaries for purposes of manipulation functions such as end-to-end error detecting codes or moving into application address space.

At the same time, the ADU now becomes the unit of error recovery. Since the ADU is defined to be the smallest unit which the application (or presentation conversion function) can deal with out of order, it follows that if even part of an ADU is lost in transmission, the application will, in general, be unable to deal with it. Since our application layer takes on the responsibility of recovering lost data, it will almost certainly need to assume the whole ADU is lost, even if parts exist.[10] Unless the presentation layer can translate the identity of the lost data into terms the application understands, the application cannot understand which of its elements have actually been lost.

This suggests that ADU lengths should be reasonably bounded, so that when data is lost the application need do no more work than necessary. Indeed, since the loss of even one bit will trigger the loss of a whole ADU, excessively large ADUs might prevent useful progress at all, since the probability of any ADU having at least one uncorrected error would approach one.

However, there will be a minimum unit, based on the details of the application data, below which the application cannot break the data and still deal with parts lost or out of order. If this minimum natural size of the ADU is too large to provide a practical unit of error free transmission, then it will be necessary to define an artificial set of subunits into which an ADU is broken for error recovery. Although this partitioning may be provided by an independent protocol module, the overall responsibility for retransmission must rest with the application.

The Architecture of Presentation Conversion

To express the processing of ADUs in a more abstract way, we note that an ADU exists in several representations or "syntaxes." Each application understands the ADU in its

[10] We assume that ADUs may be broken into smaller units suitable for transmission across physical links. Of course, lower layer recovery schemes, such as forward error correction (FEC), may be applied to these transmission units. Similarly, our general assertion regarding applications is not meant to preclude the use of ADU-level FEC.

own "local syntax." The peer applications share a common view of the ADU in some "abstract syntax." The various data transformations on the sending end (most obviously presentation conversion) convert the ADU to some "transfer syntax," and the receiving end performs the reverse transformation. Each of these syntactic forms can be viewed as a name-space in which data elements within the ADU can be identified. The abstract syntax will represent the data as some application-level data structure; the transmission syntax will probably represent the data as a numbered sequence of bytes, as transport protocols do today.

For the receiving application to deal with the ADU out of order, it is not sufficient that the receiving end understand what presentation conversion to perform. The receiving end needs to understand where to put the data that results from the conversion. In the case of file transfer, for example, this implies that, for each ADU, the sender must provide information as to its eventual location within the receiver's file.

It is not obvious how, using today's models of presentation, the sender can compute this information in order to provide it to the receiver. Traditionally, some intermediate or "transfer" representation is used for the data, and the sender and receiver do not exchange details concerning their "local" representations. If the source and the destination are unaware of the presentation conversion used by the other, then there is no guidance that the source can give the destination as to the eventual size of each ADU. So if an ADU is received out of order, there is no way to guess how large the intervening ADUs are, and thus no way to guess where in the eventual file the current ADU will go. While the receiver can perform the presentation conversion for each ADU, it must buffer it, and thereby clog the presentation pipeline.

Thus, for a general external representation such as ASN.1, there may be no way to allow ADUs to be totally processed out of order at the receiver. As an alternative, the sender and receiver can negotiate to translate in one step from the sender to the receiver's format. If this is done at the sender, then the sender can label each ADU with its location in the sender's file (i.e. before conversion), and a separate location which it will occupy in the receivers file. Using this information, the receiver can copy the data into the file at the correct location, even though intervening ADUs are missing.

In fact, the example of file transfer is too restricted to suggest the most general form of this principle. A very different application example is stream data such as video. In this case, each ADU must be identified with its location, both in space (where on the screen it goes) and in time (which video frame it is a part of). In order for the sender to compute this, it is not necessary to convert the data to the representation of the receiver; it is just necessary for the sender and receiver to agree on some higher-level name-space in which ADUs are named. The sender need only be able to compute the receiver's name for each ADU in this name-space.

Thus the more general architectural principle of presentation conversion is that the sender must perform at least enough of the conversion to be able to compute, in terms meaningful to the receiver, where or when the ADU is to be delivered. In some cases, this may imply that the sender must do all the conversion; in other situations, the receiver can do further processing as an implementation decision. Always, the sender must be able to specify the disposition of the ADU in terms meaningful to the receiver.

The final characterization of an ADU is thus the following. The sending and receiving application must define what data goes in an ADU such that

- the sender can compute a name for each ADU that permits the receiver to understand its place in the sequence of ADUs produced by the sender, and
- the sender uses a transfer syntax that permits the ADU to be processed out of order.

6 Integrated Layer Processing

Layered protocol suites provide isolation between the functional modules of distinct layers. A major architectural benefit of isolation is that it facilitates the implementation of subsystems whose scope is restricted to a small subset of the suite's layers. For example, the implementation of a network layer relay is largely independent of the upper layer protocols used by its clients.

However, we are frequently concerned with the implementation of complete end systems that coincidentally terminate most of the layers of the protocol stack. The naive implementation of a layered suite involves the sequential processing of each unit of information, as it is passed down through the individual layer entities of the transmitter's protocol stack, and as it is passed up through the peer entities of the receiver's stack. Such a simple ordering of operations may well conflict with the efficient engineering of the end systems, and it is sometimes possible to arrange the operations in a more efficient order that achieves the same result.

Ordering Constraints

Layered engineering designs should not be thought of as fundamental, but only as one approach, which must be evaluated on the basis of overhead and simplicity against other designs. As an alternative, we introduce an engineering principle called Integrated Layer Processing, or ILP, which captures the idea that the protocol be so structured as to permit the implementor the option of performing all the manipulation steps in one or two integrated processing loops, instead of performing them serially as is most often done today.

Unfortunately, traditional protocol suites often impose "precedence" or "ordering" constraints that limit the opportunities for such optimization. In particular, the shuffling of operations associated with different layers is often restricted. To admit ILP, a protocol architecture must be organized so that the interactions between processing steps, both control and data manipulation, do not interfere with their integration. We conclude that our new protocol architecture should minimize the inter-layer ordering constraints imposed on its implementors. Although we favor the ILP approach described above, a "reduced constraint architecture" would also facilitate pipelined implementations, such as the interface design described in [11]. Thus an ILP-compatible architecture preserves the benefits of isolation, while increas-

ing the range of implementation options available to end system designers.

One example of inter-layer optimization is the provision for encryption processing in the system described in [15]. The Autonet protocol suite has been carefully structured to facilitate the implementation of end system interfaces that entwine the session-specific encryption operations with data link level operations. This relaxation of the conventional ordering constraints does not interfere with layer isolation: intermediate nodes perform the traditional data link and network layer operations without regard to the session encryption operation.

To understand how interactions among steps interfere with integration, one must look at the receiving end of the connection. When sending, the software knows the complete state of the transfer, so that the manipulations can be performed independent of the control steps. When receiving data, the protocol must perform certain of the control actions in order to permit the manipulation. The details of the protocol architecture, for example the layering, will determine the ordering constraints, i.e., which manipulation steps can only be done after specific control steps. These details determine the implementation options and, thus, the efficiency of the manipulation steps. This point is the key to effective design and implementation of protocol suites.

One example of an ordering constraint is that at least some part of the data must be extracted from the network before it can be demultiplexed. It is very hard to combine the extraction and serial-to-parallel conversion step with other manipulation, except perhaps error detection, since most manipulations require the local state information, which is only identified through demultiplexing. In general, multiplexing at a given layer may impose ordering constraints, or at least complicate the re-ordering of operations across layer boundaries.[11]

Another example, discussed above, is that many manipulations can only be performed once the data unit is in order. This is true of most error detection checksums, of many encryption schemes, of most presentation transformations and of moving the data to or from the user address space. Thus, the protocol must insure that the data is in order, at least within a certain range, before performing these manipulations.

ILP Performance

This paper has presented two significant cases to consider in exploring the performance impact of ILP. In one case, there is no presentation layer; in the other, there is.

If there is no presentation conversion, then state-of-the-art implementations are already running into the performance limit imposed by the very simple data transformations of the transport and lower layers – data copying and checksums. In these cases, ILP may be a very practical next step to improve throughput, especially with RISC processors. This will, of course, not be true universally; ILP is just an engineering principle, to be applied only when useful. But the very simple performance experiments we reported above, as well as some of the advanced experiments performed by Van Jacobson on TCP [3], suggest the utility of this approach.

If there is a presentation conversion being performed, then the cost of this step may well dominate the other manipulation costs. In this case, the application code, which must participate in the presentation conversion, must be the focus of performance tuning. ILP can help here, because the processing steps can be pipelined in a flexible manner.

Up to this point, we have introduced a number of key problems and concepts – the problem of reordering, the requirement for efficient pipelining of the presentation conversion, and so on. An overall structure that takes these concepts into account is based on two manipulation stages.

First, the transmission data units are received from the network. They are then examined to determine which ADU they belong to (the demultiplexing control operation) and where in the ADU they go (the re-ordering control operation). If part of an ADU is lost, then either some lower layer may try to fix the problem (if it has buffered sufficient information), the receiving application may cope, or the sending application is instructed to resend the whole ADU.

Once a complete ADU is received, even if it is out of order with respect to other ADUs in the same application association, it can be passed to the application for the second stage of processing. This processing will include all the required data manipulations, including error and encryption checks, and possibly presentation conversion, if the sender has not performed this task completely. The two stage approach is justified by the distinction between two classes of "out of order" events: misordering of transmission units; and misordering of complete ADUs. In the normal case where all transmission units arrive in order, the two stages may be fully integrated.

Even if the data does not require presentation conversion, it will require dispatching to the proper locations in memory. In some cases, this may be simple, if the data is destined for the file system as a sequence of linear disk blocks. A more general case will require that the data in the ADU be separated into different values which are stored in different variables of some program. This is the general paradigm of the Remote Procedure Call, in which the incoming data is made to appear as parameters of a subroutine call in some high level programming language.

This requirement to copy the data into locations that are part of the application address space, and which may be distributed in that address space rather than being a linear region, is a critical architectural constraint. One proposal for speeding up protocols is to perform processing on a specialized outboard processor. We assert that it will prove too complex to provide a specialized processor with all the information necessary for it to copy the data properly into the application address space.[12] While in some cases, such as raw file transfer, it might be easy, in general it would require giving to the outboard processor information of the same bulk and complexity as the incoming data itself. For this reason, most proposals for outboard processors do not include the presentation layer in the tasks to be performed outboard.

[11] The case against layered multiplexing is presented in [18, 12].

[12] A more realistic proposition is the partitioning of application and presentation processing across the general purpose nodes of a multiprocessor system.

The key argument in favor of ILP is that an integrated processing loop is more efficient than several separate steps which read the data from memory, possibly convert it, and write it again. This point seems most obviously true for RISC chips, where a major performance limitation is memory cycles. We claim that layered engineering designs should not be thought of as fundamental, but only as one approach, which must be evaluated on the basis of overhead and simplicity against other designs. ILP designs may lead to greater efficiency through careful attention to the ordering of the processing steps and the delineation of the data units.

7 Application Level Framing Revisited

The concept of Application Level Framing was a key to achieving Integrated Layer Processing, because it provided a single common unit of data, the ADU, over which manipulation was defined. In contrast, protocols today define some operations over the packet, and others over application boundaries.

We believe that once the concept of the ADU is understood, it will provide a more natural and direct way to deal with a variety of protocol problems. As a particular example, consider the problem of connecting networks to a parallel processor.

A network is usually thought of as a serial device. This sort of device does not match the nature of some parallel processors. One of the design goals of a parallel processor is to avoid building any one hot spot in the architecture which must run at the aggregate speed of the total processor. But lacking such a spot, there is no place to connect a high-speed serial network. The solution seems to be to separate the network into several parts, each of which delivers part of the data to part of the processor. But how is the data to be dispatched to the correct part? If the data is sent to the parallel processor using a traditional protocol such as TCP, there is no way the transport can understand the structure of the incoming data. However, if the data is organized into ADUs, each ADU will contain enough information to control its own delivery.

In summary, the next generation of protocol must have a very general model of the application. Not only must the network support traditional computers, but other devices, including video sources and sinks, specialized memories, and so on. Since the ADU delivery information is not just visible at the "application protocol layer" but to all the protocol functions, it can be used to permit a wide variety of implementation approaches for various sorts of devices.

8 Conclusions

In this paper, we make several structural observations about protocols, and attempt to draw some architectural conclusions. In summary, our reasoning has the following points.

- Data manipulation costs more than transfer control operations.
- Presentation can cost more than all other manipulations combined.
- To implement presentation efficiently, it is necessary to keep the processing pipeline going, including the application process.
- Application data units are the natural pipelining units. They correspond to what applications want, not to the network technology of the day, which can and will change in the near future.
- The key architectural principle we have identified is Application Level Framing.
- The key engineering principle is Integrated Layer Processing, which allows applications to process their data incrementally, and permits efficient implementation of data manipulations on RISC processors.

In this approach to protocol architecture, the different functions are "next to" each other, not "on top of," to the extent possible. The traditional functions of transport and session are control functions, and as such should not have a strict relation to the manipulation functions such as presentation conversion. This is a more parallel organizational model, similar to the HOPS model of Haas [7].

This discussion of protocol structure should not be taken to mean that layering is unsuited as a tool for protocol design. The principle role of layering is the semantic isolation of functional modules. Although alternative organizational schemes exist, layered isolation has proven to be particularly effective in the network environment.

At the Network layer and below, intermediate relay entities must participate in some aspects of the communications process. Within the context of a layered architecture, intermediate entities can operate at one or more layers without regard to the semantic content of the symbols being exchanged at the upper or lower layers. This constitutes a powerful argument in favor of layered isolation.

However, the same reasoning does not apply at Transport and above, where symbols are exchanged on an end-to-end basis. Alternative organizational schemes may be better suited to the functional decomposition of what are presently referred to as the upper layer functions. In particular, we observe that many of the Transport, Session, and Presentation functions can be structured in a less constrained manner.

A potential drawback to the ILP approach is that it could lead to complex designs in which each protocol stack variant has its own fully customized implementation. Clearly this would complicate the maintainability and overall utility of the protocol software. However, we believe that there are approaches to protocol organization that can minimize the liabilities associated with "vertical integration."

In existing protocol suites the semantics of a functional module, i.e., a layer protocol, are closely tied to the syntax of the symbols that are exchanged. Usually there is a one-to-one correspondence between semantic information and specific fields within protocol data units. This arrangement gives rise to the present day scheme of hierarchical encapsulation in which each layer appends its own syntactical symbols to those generated by the layers above.

We propose that the semantics of a functional module be decoupled from the syntax used to effect the exchange of protocol control information. A single syntactical field could be interpreted by a number of modules, with each applying its own semantic rules as appropriate. Thus, where layering is appropriate, semantic isolation can be retained without the present day overhead of layered encapsulation and framing.[13]

In many respects this approach corresponds to the "compilation" of the protocol suite, while the encapsulation approach corresponds to its "interpretation". We believe that the principle of Application Layer Framing permits this shared use of fields and structures across layers, and ILP is one speculative example of a "compiled" implementation of a protocol suite.

9 Acknowledgements

James Davin performed the presentation conversion experiments described in section four. Karen Sollins provided considerable assistance in detailed discussions concerning the key principles and in the alignment of the sometimes divergent views of the co-authors. John Wroclawski, Karen Sollins, Chuck Davin, and the referees provided criticism of earlier drafts which substantially improved the final version of this paper.

In many respects, this paper represents the blending of insights gained from our past activities, both at MIT and the University of Cambridge. The ideas developed in this paper were also extensively motivated by discussions with the End to End Research Group of the Internet Research Task Force, in particular the comments of Van Jacobson.

References

[1] E.A. Arnould et al. The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers. In *ASPLOS-III Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, IEEE/ACM, April 1989.

[2] G. Chesson, B. Eich, V. Schryver, A. Cherenson, and A. Whaley. *XTP Protocol Definition.* Technical Report Revision 3.0, Silicon Graphics, Inc., January 1988.

[3] D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *IEEE Communications*, 27(6):23–29, June 1989.

[4] David Clark. The Structuring of Systems Using Upcalls. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 171–180, Association for Computing Machinery, Oakland, CA, December 1985.

[5] A. G. Fraser and W. T. Marshall. Data Transport in a Byte Stream Network. *IEEE Journal on Selected Areas in Communications*, 7(7):1020–1033, September 1989.

[6] Rainer Handel. Evolution of ISDN Towards Broadband ISDN. *IEEE Network*, 3(1):7–13, January 1989.

[13] Datakit control byte codes [5] have these properties.

[7] Zygmunt Hass. A Communication Architecture for High-speed Networking. In preparation.

[8] C. Huitema and A. Doghri. A High Speed Approach for the OSI Presentation Protocol. In H. Rudin and R. Williamson, editors, *Protocols for High-Speed Networks*, Elsevier Science Publishers, May 1989. IFIP.

[9] ISO. *Information Processing Systems – Open Systems Interconnection – Basic Reference Model.* 1984. ISO-7498.

[10] ISO. *Information Processing Systems – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1).* International Standard ISO-8824.

[11] J. Kanakia and D.R. Cheriton. *The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors.* Technical Report, Stanford University, Stanford, CA, November 1987.

[12] D. R. McAuley. *Protocol Design for High Speed Networks.* Technical Report TR No. 186, University of Cambridge Computer Laboratory, January 1990.

[13] J. Postel and J. Reynolds. Telnet Protocol Specification NIC-RFC-854. *DDN Protocol Handbook*, 2:2.575–2.589, May 1983.

[14] K. Sabnani and A. Netravali. A High Speed Transport Protocol for Datagram / Virtual Circuit Networks. In *ACM Sigcomm '89*, pages 146–157, Association for Computing Machinery, Austin, Texas, September 1989.

[15] M.D. Schroeder et al. *Autonet: A High-speed, Self-configuring Local Area Network Using Point-to-Point Links.* Technical Report Research Report 59, Digital Systems Research Center, April 1990.

[16] Sun Microsystems, Inc. *XDR: External Data Representation Standard.* RFC - 1014, Network Information Center, SRI International, June 1987.

[17] L. Svobodova. Implementing OSI Systems. *IEEE Journal on Selected Areas in Communications*, 7(7):1115–1130, September 1989.

[18] D. L. Tennenhouse. Layered Multiplexing Considered Harmful. In H. Rudin and R. Williamson, editors, *Protocols for High-Speed Networks*, Elsevier Science Publishers, May 1989. IFIP.

[19] CCITT Study Group XVIII. *Draft 1990 B-ISDN Recommendations.* January 1990.