M.I.T. Laboratory for Computer Science

Request for Comments No. 214
December 4, 1981

An Argument for Soft Layering of Protocols

by Geoffrey Cooper

The following is a Master's thesis proposal.

1. Introduction

The technique of protocol layering has lately become a matter of contention. While the merits of a layered design and implementation are clear, experience with the implementation of layered protocols has cast doubts upon the potential for an efficient implementation of layered designs. This thesis will examine whether a particular extension to the technique of protocol layering can decrease the implementation cost associated with layered design.

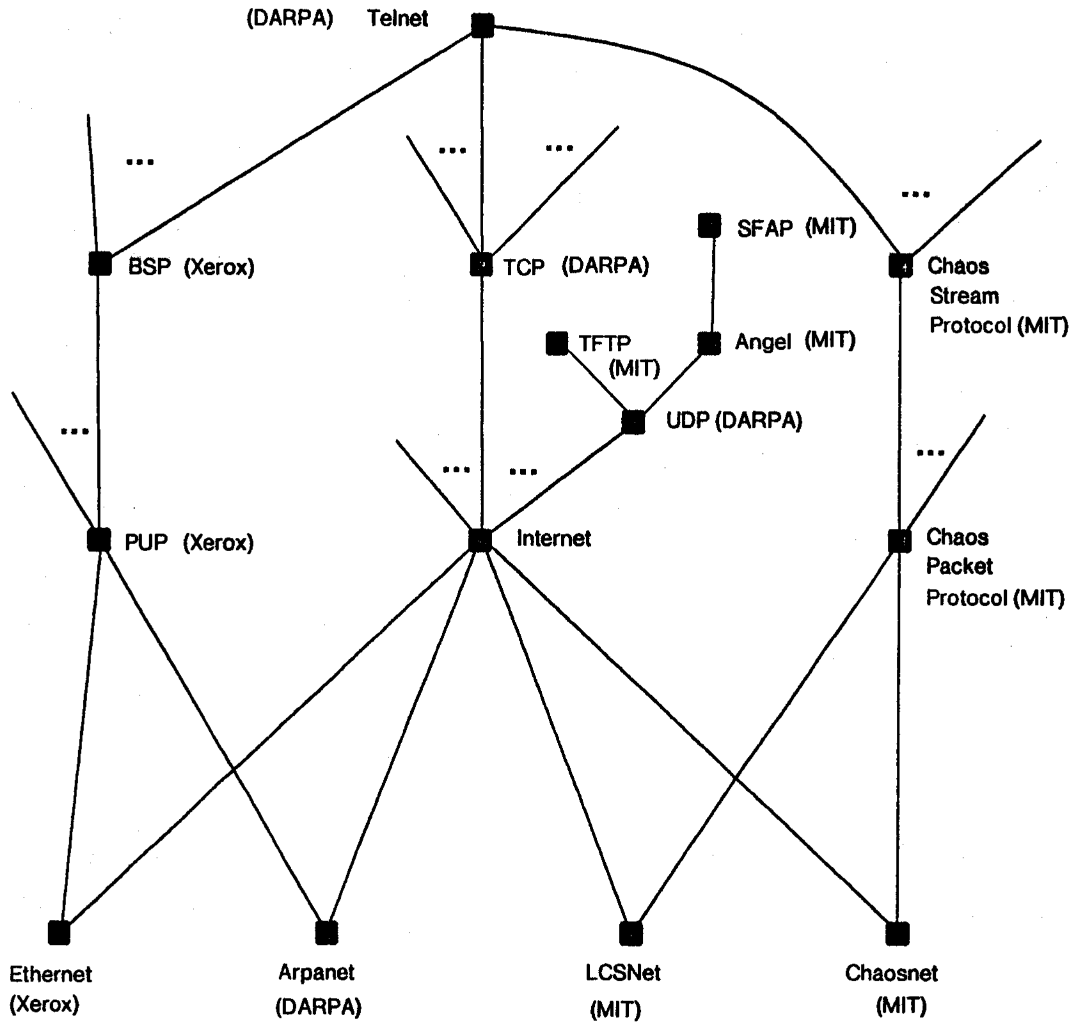
2. Layered Protocols

A protocol is a mechanism by which cooperating processes interact. To allow a higher degree of functional modularity, protocols are arranged in a *layered* structure, such that each protocol hides the layer directly beneath it from the layer directly above it. An example of this is the DoD standard set of protocols [Postel 80a]. The way in which layered protocols interact is illustrated by arranging the protocols on a lattice, as in figure 1. In this representation, protocols which are at a higher logical level are placed in a higher position on the lattice.

Each protocol presents an interface to the layer above it. This interface is equivalent to the "fire walls" of a ring protection scheme [Saltzer 74]. Each protocol also has a model of the layer directly below it. This "model" is often no more than a protocol specification – when a protocol may only lie above one specific protocol (as is the case with TCP) – but it might also represent a minimal set of common characteristics that protocols beneath it must have, as in the Internet protocol [Postel 81], and the Telnet protocol [Postel 80b].

It has been suggested that protocols be defined as particular abstractions of a data communications link. This definition suggests a parallel between protocols and computer languages, which is useful to keep in mind. Unfortunately, the logical extension of this definition to layered protocols fails to allow for some

Figure 1: Layered Protocols, Arranged on a "layered-lattice"



common situations in protocol design.[1] For this reason, we prefer the more basic definition given above.

3. Problems with Protocol Layering

The greatest advantage of a layered design of protocols is the "plug compatibility" of different implementation of layers, and of layers one to the other. In principle, a new implementation of a layer may replace the old with no changes to lower layers, and a small number of changes to higher layers (or none, if the interface is the kept the same in both directions). Yet this comes only at a cost. With each added layer, a protocol implementation will become less efficient.

One way in which this happens is the added number of levels of procedure or system calls needed to

[1]Example: The Gateway-to-Gateway (GGP) [Strazisar 79] protocol makes use of the Internet protocol to route send packets through internetwork connections. Simultaneously, the Internet protocol uses the information that the GGP protocol determines to do this routing. Depending on how one views the situation, GGP is either a refining abstraction on the Internet protocol, or an abstraction which the Internet protocol refines; it would lie both above and below the Internet layer.

implement an extra layer of software. Cohen, Postel [Cohen 79] and others have pointed out that where this is a problem, layers may be compressed in an implementation to achieve the desired efficiency, albeit at a cost of the maintainability of a particular implementation. It is easy to see how special hardware (e.g. microcode) can be used to further reduce this effect.

Of more concern is the loss of efficiency due to a lack of communication between layers. There are two ways in which this problem shows itself. They are listed below:

The Asynchrony Problem:

Interacting processes are mutually asynchronous, so an asynchronous I/O device must be handled by some layer of a layered protocol scheme. It is common for this layer to hide some of the asynchronous aspects of communications from layers above it, to simplify their design and implementation. This means that the different layers of protocol are potentially asynchronous, yet must cooperate to send data. The central problem that arises is how to deal with the case where one layer wishes to send data but the other(s) do not. This is usually approached by allowing each layer to send data at any time, with the proviso that all data need not be forwarded to or acted upon by every layer. The result is often a duplication of effort, since layers might not realize that data need be sent in time to encode their data into other layers' packets.

In TCP-Telnet, for example, TCP may send packets, for the purposes of recovering and sequencing data, which are never passed to Telnet. When Telnet passes a character to TCP, TCP must transmit that character and make sure that it was delivered correctly. This task requires at least one additional packet. End-to-end arguments indicate that Telnet or some protocol above it (in this case, a person sitting in front of a terminal) must verify for itself that the data was correctly transferred and acted upon, so at some point, the foreign Telnet will also send a response. If the foreign TCP is lucky, the foreign Telnet's response to the data can be "piggy-backed" in the same packet as the foreign TCP's verification of the first data. If not, an extra packet will need be sent.

The Timeout Problem:

Because of end-to-end arguments, each layer which maintains any connection state must set timeouts of its own. In an Internet-TCP-Telnet connection, three levels of timeouts are needed. The Internet layer must set timeouts when reassembling packets, the TCP layer to determine if the foreign host is still alive, and the Telnet layer to determine if the foreign Telnet layer is accepting the data correctly (although the latter is usually left out, by relying on the human user of the Telnet program to determine when this has occurred). Often many of these timers are redundant, and slow down operations by causing unnecessary process scheduling.

4. An Extension to Conventional Layering

Each of the above problems rests in some way on so-called *end-to-end* arguments, as described in [Saltzer 80]. End-to-End arguments push the responsibility for checking the veracity and sequence of transmitted data to the uppermost protocol. Lower level protocols can guarantee no more than correct delivery of data.

wrong → ref!

A way of increasing efficiency in a layered environment is to use end-to-end arguments to advantage. If the highest level must perform certain checks on the data, it should be possible to avoid doing what amount to redundant checks at lower levels. The idea is to "soften" the boundaries between protocol layers so that each layer has a better idea of the operations of the layers above and below it. We refer to this technique as *soft protocol layering.*

In this thesis, protocol specifications will be extended to allow for soft layering. Specifically, a protocol specification will consist of

1. A description of the protocol (as before)
2. A model of the layer beneath the protocol (as before)
3. A model of the layer above the protocol (new)

For a protocol to operate correctly (i.e., for data to be transferred in the desired fashion), only the first two parts of the specification need be used. When the third part of protocol matches the application as well, greater efficiency will result.

The manner in which soft layering deals with the problems of the previous section is straightforward. Lower levels of protocol have some understanding of the nature of the data that they are passing to higher levels, so lower levels will often know when a higher level response will be forthcoming. In this case, lower level processing can be deferred until this data arrives, so that a single packet may be sent. This solves the asynchrony problem.

The timeout problem is solved by having the different levels of protocol cooperate on determining what timeouts are necessary, so that only a single timeout can be sent. The main utility in this approach is that it allows a layer to hand the responsibility for setting timeouts to a different layer. For example, if a "carrier protocol" (such as TCP) is implemented in the kernel of an operating system, it should be possible for a process using TCP to block waiting for data, relying on the lower level protocol to wake up the process either when data arrives, or upon determining that there is an exceptional condition that requires the attention of the higher level protocol.

Much of the work to be done for the thesis will consist of the design and implementation of a protocol called *Angel* [Cooper 81], which supports soft layering in its specification as described above.

Angel provides a reliable message service. As described above, it has a model of the higher level protocol that calls it. The higher level protocol, known as the *letter-level*, connects two processes for the sending of higher-level messages called *letters*. A letter is a typed, tagged block of data, and the sequence of transmitted letters is ordered. It is assumed that letters are never retransmitted at the letter level, since reliable delivery is assured by Angel.

A letter may elicit a letter in response, although responses need not arrive in the same order as the letters to which they respond. For example, if the letter-level protocol is a file access protocol such as SFAP ([Cooper 80], see below), four letters might be sent, eliciting four responses from the foreign letter level:

Initial Letter	*Contents of Responding Letter*
readBlock(1)	<Block 1>
readBlock(15)	<Block 15>
readBlock(2)	<Block 2>
readBlock(3)	<Block 3>

Let us assume that all four responses arrive at the server before any action is taken, as might be the case with a large computer system, where process wake ups are expensive. The file access server would probably benefit from reading the blocks from secondary storage in disk order, rather than letter arrival order. Since angel allows for letter-level responses to arrive out of order, the blocks may be read by the server in whatever manner is most efficient, and can be sent over the network in the order they are read.

The interface between angel and the letter-level protocol is based on angel's model of the letter-level. When a letter is sent, angel is informed whether the letter is a response to another letter, and whether the letter will, in the normal course of events, elicit a response from the foreign letter-level protocol. When a letter arrives, angel can determine whether the letter is a response to a letter sent earlier. If a letter is sent, with a response expected, and the response does not arrive, angel will timeout and report this failure to the letter-level. This solves the timeout problem, as described above, by allowing the letter-level to delegate the setting of timeouts to angel. Since all timeouts are set at the same level, timers which are redundant may be eliminated.

5. Work for the Thesis

The following projects will form part of this thesis:

1. The Design of the Angel Protocol

2. An Angel implementation on an Alto computer in BCPL

3. A version of TFTP, modified for use with angel, on an Alto computer in BCPL

4. The Design of the SFAP II Protocol (SFAP: Simple File Access Protocol), a second version of the SFAP protocol [Cooper 80], suitable for use with Angel.

5. The implementation of SFAP Server and User software on an Alto computer, in BCPL.

The purpose of the implementations described above is to demonstrate that Angel can be implemented in a way that allows the transfer of data at high speeds. A file transfer protocol implementation demonstrates this in case where the amount of data to be transferred is known well in advance.

In SFAP, data is transferred in more random fashion, as negotiations take place between the cooperating processes. The efficient implementation of SFAP on top of the same Angel implementation that efficiently implemented a file transfer will demonstrate that soft layering has not severely limited the range of different higher level protocols that are suitable for use with Angel.

The implementations will be carried out on an Alto personal computer. Substantial network software already exists for this machine, so reasonable comparisons between the efficiency of Angel and that of other layered protocols will be possible.

The thesis is scheduled for completion in January, 1982.

References

[Cohen 79]
Danny Cohen & Jonathan B. Postel.
On Protocol Multiplexing.
Proceedings of the sixth Data Communications Symposium , November, 1979.

[Cooper 80]
Geoffrey Cooper.
SFAP: A Simple File Access Protocol.
Technical Report NIN-29, MIT-CSR, July, 1980.

[Cooper 81]
Geoffrey Cooper.
The Angel Protocol.
Technical Report RFC-213, MIT-CSC, December, 1981.

[Postel 80a]
J. Postel.
Internet Protocol Handbook.
Technical Report RFC 774, USC-ISI, October, 1980.

[Postel 80b]
J. Postel.
Telnet Protocol Specification.
Technical Report RFC 764/IEN 148, USC-ISI, June, 1980.

[Postel 81]
J. Postel.
DoD Standard Internet Protocol.
Technical Report RFC 791, DARPA, September, 1981.

[Saltzer 74]
Jerome H. Saltzer.
Protection and the Control of Information Sharing in Multics.
Communications of the ACM , July, 1974.

[Saltzer 80]
J. H. Saltzer.
End-to-End Arguments in System Design.
Technical Report CSR-RFC 185, MIT-CSR, April, 1980.

[Strazisar 79]
Virginia Strazisar.
How to Build a Gateway.
Technical Report IEN 109, BBN, August, 1979.