Concept of fate-sharing and soft state

THE DESIGN PHILOSOPHY OF THE DARPA INTERNET PROTOCOLS

David D. Clark

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Ma. 02139

Abstract

~1973

The Internet protocol suite, TCP/IP, was first proposed fifteen years ago. It was developed by the Defense Advanced Research Projects Agency (DARPA), and has been used widely in military and commercial systems. While there have been papers and specifications that describe how the protocols work, it is sometimes difficult to deduce from these why the protocol is as it is. For example, the Internet protocol is based on a connectionless or datagram mode of service. The motivation for this has been greatly misunderstood. This paper attempts to capture some of the early reasoning which shaped the Internet protocols.

1. Introduction

For the last 15 years[1], the Advanced Research Projects Agency of the U.S. Department of Defense has been developing a suite of protocols for packet switched networking. These protocols, which include the Internet Protocol (IP), and the Transmission Control Protocol (TCP), are now U.S. Department of Defense standards for internetworking, and are in wide use in the commercial networking environment. The ideas developed in this effort have also influenced other protocol suites, most importantly the connectionless configuration of the ISO protocols[2, 3, 4].

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. N00014-83-K-0125.

While specific information on the DOD protocols is fairly generally available[5, 6, 7], it is sometimes difficult to determine the motivation and reasoning which led to the design.

In fact, the design philosophy has evolved considerably from the first proposal to the current standards. For example, the idea of the datagram, or connectionless service, does not receive particular emphasis in the first paper, but has come to be the defining characteristic of the protocol. Another example is the layering of the architecture into the IP and TCP layers. This seems basic to the design, but was also not a part of the original proposal. These changes in the Internet design arose through the repeated pattern of implementation and testing that occurred before the standards were set.

The Internet architecture is still evolving. Sometimes a new extension challenges one of the design principles, but in any case an understanding of the history of the design provides a necessary context for current design extensions. The connectionless configuration of ISO protocols has also been colored by the history of the Internet suite, so an understanding of the Internet design philosophy may be helpful to those working with ISO.

This paper catalogs one view of the original objectives of the Internet architecture, and discusses the relation between these goals and the important features of the protocols.

2. Fundamental Goal

The top level goal for the DARPA Internet Architecture was to develop an effective technique for multiplexed utilization of existing interconnected networks. Some elaboration is appropriate to make clear the meaning of that goal.

The components of the Internet were networks, which were to be interconnected to provide some larger service. The original goal was to connect together the original ARPANET[8] with the ARPA packet radio network[9, 10], in order to give users on the packet radio network access to the large service machines on the ARPANET. At the time it was assumed that there would be other sorts of networks to interconnect, although the local area network had not yet emerged.

An alternative to interconnecting existing networks would have been to design a unified system which incorporated a variety of different transmission media, a multi-media network. While this might have permitted a higher degree of integration, and thus better performance, it was felt that it was necessary to incorporate the then existing network architectures if Internet was to be useful in a practical sense. Further, networks represent administrative boundaries of control, and it was an ambition of this project to come to grips with the problem of integrating a number of separately administrated entities into a common utility.

The technique selected for multiplexing was packet switching. An alternative such as circuit switching could have been considered, but the applications being supported, such as remote login, were naturally served by the packet switching paradigm, and the networks which were to be integrated together in this project were packet switching networks. So packet switching was accepted as a fundamental component of the Internet architecture.

The final aspect of this fundamental goal was the assumption of the particular technique for interconnecting these networks. Since the technique of store and forward packet switching, as demonstrated in the previous DARPA project, the ARPANET, was well understood, the top level assumption was that networks would be interconnected by a layer of Internet packet switches, which were called gateways.

From these assumptions comes the fundamental structure of the Internet: a packet switched communications facility in which a number of distinguishable networks are connected together using packet communications processors called gateways which implement a store and forward packet forwarding algorithm.

3. Second Level Goals

The top level goal stated in the previous section contains the word "effective," without offering any definition of what an effective interconnection must achieve. The following list summarizes a more detailed set of goals which were established for the Internet architecture.

fault tolerance

1. Internet communication must continue despite loss of networks or gateways.
2. The Internet must support multiple types of communications service.
3. The Internet architecture must accommodate a variety of networks.
4. The Internet architecture must permit distributed management of its resources.

This list of goals is in decreasing priority.

5. The Internet architecture must be cost effective.
6. The Internet architecture must permit host attachment with a low level of effort.
7. The resources used in the internet architecture must be accountable.

This set of goals might seem to be nothing more than a checklist of all the desirable network features. It is important to understand that these goals are in order of importance, and an entirely different network architecture would result if the order were changed. For example, since this network was designed to operate in a military context, which implied the possibility of a hostile environment, survivability was put as a first goal, and accountability as a last goal. During wartime, one is less concerned with detailed accounting of resources used than with mustering whatever resources are available and rapidly deploying them in an operational manner. While the architects of the Internet were mindful of accountability, the problem received very little attention during the early stages of the design, and is only now being considered. An architecture primarily for commercial deployment would clearly place these goals at the opposite end of the list.

Similarly, the goal that the architecture be cost effective is clearly on the list, but below certain other goals, such as distributed management, or support of a wide variety of networks. Other protocol suites, including some of the more popular commercial architectures, have been optimized to a particular kind of network, for example a long haul store and forward network built of medium speed telephone lines, and deliver a very cost effective solution in this context, in exchange for dealing somewhat poorly with other kinds of nets, such as local area nets.

The reader should consider carefully the above list of goals, and recognize that this is not a "motherhood" list, but a set of priorities which strongly colored the design decisions within the Internet architecture. The following sections discuss the relationship between this list and the features of the Internet.

4. Survivability in the Face of Failure

The most important goal on the list is that the Internet should continue to supply communications service, even though networks and gateways are failing. In particular, this goal was interpreted to mean that if two entities are communicating over the Internet, and some failure causes the Internet to be temporarily disrupted and reconfigured to reconstitute the service, then the entities communicating should be able to continue without having to reestablish or reset the high level state of their conversation. More concretely, at the service interface of the transport layer, this architecture provides no facility to communicate to the client of the transport service that the synchronization between the sender and the receiver may have been lost. It was an assumption in this architecture that synchronization would never be lost unless there was no physical path over which any sort of communication could be achieved. In other words, at the top of transport, there is only one failure, and it is total partition. The architecture was to mask completely any transient failure.

To achieve this goal, the state information which describes the on-going conversation must be protected. Specific examples of state information would be the number of packets transmitted, the number of packets acknowledged, or the number of outstanding flow control permissions. If the lower layers of the architecture lose this information, they will not be able to tell if data has been lost, and the application layer will have to cope with the loss of synchrony. This architecture insisted that this disruption not occur, which meant that the state information must be protected from loss.

In some network architectures, this state is stored in the intermediate packet switching nodes of the network. In this case, to protect the information from loss, it must replicated. Because of the distributed nature of the replication, algorithms to ensure robust replication are themselves difficult to build, and few networks with distributed state information provide any sort of protection against failure. The alternative, which this architecture chose, is to take this information and gather it at the endpoint of the net, at the entity which is utilizing the service of the network. I call this approach to reliability "fate-sharing." The fate-sharing model suggests that it is acceptable to lose the state information associated with an entity if, at the same time, the entity itself is lost. Specifically, information about transport level synchronization is stored in the host which is attached to the net and using its communication service.

Benefits of fate sharing:

There are two important advantages to fate-sharing over replication. First, fate-sharing protects against any number of intermediate failures, whereas replication can only protect against a certain number (less than the number of replicated copies). Second, fate-sharing is much easier to engineer than replication.

There are two consequences to the fate-sharing approach to survivability. First, the intermediate packet switching nodes, or gateways, must not have any essential state information about on-going connections. Instead, they are stateless packet switches, a class of network design sometimes called a "datagram" network. Secondly, rather more trust is placed in the host machine than in an architecture where the network ensures the reliable delivery of data. If the host resident algorithms that ensure the sequencing and acknowledgment of data fail, applications on that machine are prevented from operation.

Despite the the fact that survivability is the first goal in the list, it is still second to the top level goal of interconnection of existing networks. A more survivable technology might have resulted from a single multi-media network design. For example, the Internet makes very weak assumptions about the ability of a network to report that it has failed. Internet is thus forced to detect network failures using Internet level mechanisms, with the potential for a slower and less specific error detection.

5. Types of Service

The second goal of the Internet architecture is that it should support, at the transport service level, a variety of types of service. Different types of service are distinguished by differing requirements for such things as speed, latency and reliability. The traditional type of service is the bi-directional reliable delivery of data. This service, which is sometimes called a "virtual circuit" service, is appropriate for such applications as remote login or file transfer. It was the first service provided in the Internet architecture, using the Transmission Control Protocol (TCP)[11]. It was early recognized that even this service had multiple variants, because remote login required a service with low delay in delivery, but low requirements for bandwidth, while file transfer was less concerned with delay, but very concerned with high throughput. TCP attempted to provide both these types of service.

The initial concept of TCP was that it could be general enough to support any needed type of service. However, as the full range of needed services became clear, it seemed too difficult to build support for all of them into one protocol.

The first example of a service outside the range of TCP was support for XNET[12], the cross-Internet debugger. TCP did not seem a suitable transport for XNET for several reasons. First, a debugger protocol should not be reliable. This conclusion may seem odd, but under conditions of stress or failure (which may be exactly when a debugger is needed) asking for reliable communications may prevent any communications at all. It is much better to build a service which can deal with whatever gets through, rather than insisting that every byte sent be delivered in order. Second, if TCP is general enough to deal with a broad range of clients, it is presumably somewhat complex. Again, it seemed wrong to expect support for this complexity in a debugging environment, which may lack even basic services expected in an operating system (e.g. support for timers.) So XNET was designed to run directly on top of the datagram service provided by Internet.

Example of debugger

Another service which did not fit TCP was real time delivery of digitized speech, which was needed to support the teleconferencing aspect of command and control applications. In real time digital speech, the primary requirement is not a reliable service, but a service which minimizes and smooths the delay in the delivery of packets. The application layer is digitizing the analog speech, packetizing the resulting bits, and sending them out across the network on a regular basis. They must arrive at the receiver at a regular basis in order to be converted back to the analog signal. If packets do not arrive when expected, it is impossible to reassemble the signal in real time. A surprising observation about the control of variation in delay is that the most serious source of delay in networks is the mechanism to provide reliable delivery. A typical reliable transport protocol responds to a missing packet by requesting a retransmission and delaying the delivery of any subsequent packets until the lost packet has been retransmitted. It then delivers that packet and all remaining ones in sequence. The delay while this occurs can be many times the round trip delivery time of the net, and may completely disrupt the speech reassembly algorithm. In contrast, it is very easy to cope with an occasional missing packet. The missing speech can simply be replaced by a short period of silence, which in most cases does not impair the intelligibility of the speech to the listening human. If it does, high level error correction can occur, and the listener can ask the speaker to repeat the damaged phrase.

It was thus decided, fairly early in the development of the Internet architecture, that more than one transport service would be required, and the architecture must be prepared to tolerate simultaneously transports which wish to constrain reliability, delay, or bandwidth, at a minimum.

This goal caused TCP and IP, which originally had been a single protocol in the architecture, to be separated into two layers. TCP provided one particular type of service, the reliable sequenced data stream, while IP attempted to provide a basic building block out of which a variety of types of service could be built. This building block was the datagram, which had also been adopted to support survivability. Since the reliability associated with the delivery of a datagram was not guaranteed, but "best effort," it was possible to build out of the datagram a service that was reliable (by acknowledging and retransmitting at a higher level), or a service which traded reliability for the primitive delay characteristics of the underlying network substrate. The User Datagram Protocol (UDP)[13] was created to provide a application-level interface to the basic datagram service of Internet.

The architecture did not wish to assume that the underlying networks themselves support multiple types of services, because this would violate the goal of using existing networks. Instead, the hope was that multiple types of service could be constructed out of the basic datagram building block using algorithms within the host and the gateway. For example, (although this is not done in most current implementations) it is possible to take datagrams which are associated with a controlled delay but unreliable service and place them at the head of the transmission queues unless their lifetime has expired, in which case they would be discarded; while packets associated with reliable streams would be placed at the back of the queues, but never discarded, no matter how long they had been in the net.

It proved more difficult than first hoped to provide multiple types of service without explicit support from the underlying networks. The most serious problem was that networks designed with one particular type of service in mind were not flexible enough to support other services. Most commonly, a network will have been designed under the assumption that it should deliver reliable service, and will inject delays as a part of producing reliable service, whether or not this reliability is desired. The interface behavior defined by X.25, for example, implies reliable delivery, and there is no way to turn this feature off. Therefore, although Internet operates successfully over X.25 networks it cannot deliver the desired variability of type service in that context. Other networks which have an intrinsic datagram service are much more flexible in the type of service they will permit, but these networks are much less common, especially in the long-haul context.

6. Varieties of Networks

It was very important for the success of the Internet architecture that it be able to incorporate and utilize a wide variety of network technologies, including military and commercial facilities. The Internet architecture has been very successful in meeting this goal: it is operated over a wide variety of networks, including long haul nets (the ARPANET itself and various X.25 networks), local area nets (Ethernet, ringnet, etc.), broadcast satellite nets (the DARPA Atlantic Satellite Network[14, 15] operating at 64 kilobits per second and the DARPA Experimental Wideband Satellite Net,[16] operating within the United States at 3 megabits per second), packet radio networks (the DARPA packet radio network, as well as an experimental British packet radio net and a network developed by amateur radio operators), a variety of serial links, ranging from 1200 bit per second asynchronous connections to T1 links, and a variety of other ad hoc facilities, including intercomputer busses and the transport service provided by the higher layers of other network suites, such as IBM's HASP.

The Internet architecture achieves this flexibility by making a minimum set of assumptions about the function which the net will provide. The basic assumption is that network can transport a packet or datagram. The packet must be of reasonable size, perhaps 100 bytes minimum, and should be delivered with reasonable but not perfect reliability. The network must have some suitable form of addressing if it is more than a point to point link.

There are a number of services which are explicitly not assumed from the network. These include reliable or sequenced delivery, network level broadcast or multicast, priority ranking of transmitted packet, support for multiple types of service, and internal knowledge of failures, speeds, or delays. If these services had been required, then in order to accommodate a network within the Internet, it would be necessary either that the network support these services directly, or that the network interface software provide enhancements to simulate these services at the endpoint of the network. It was felt that this was an undesirable approach, because these services would have to be re-engineered and reimplemented for every single network and every single host interface to every network. By engineering these services at the transport, for example reliable delivery via TCP, the engineering must be done only once, and the implementation must be done only once for each host. After that, the implementation of interface software for a new network is usually very simple.

7. Other Goals

The three goals discussed so far were those which had the most profound impact on the design on the architecture. The remaining goals, because they were lower in importance, were perhaps less effectively met, or not so completely engineered. The goal of permitting distributed management of the Internet has certainly been met in certain respects. For example, not all of the gateways in the Internet are implemented and managed by the same agency. There are several different management centers within the deployed Internet, each operating a subset of the gateways, and there is a two-tiered routing algorithm which permits gateways from different administrations to exchange routing tables, even though they do not completely trust each other, and a variety of private routing algorithms used among the gateways in a single administration. Similarly, the various organizations which manage the gateways are not necessarily the same organizations that manage the networks to which the gateways are attached.

On the other hand, some of the most significant problems with the Internet today relate to lack of sufficient tools for distributed management, especially in the area of routing. In the large internet being currently operated, routing decisions need to be constrained by policies for resource usage. Today this can be done only in a very limited way, which requires manual setting of tables. This is error-prone and at the same time not sufficiently powerful. The most important change in the Internet architecture over the next few years will probably be the development of a new generation of tools for management of resources in the context of multiple administrations.

It is clear that in certain circumstances, the Internet architecture does not produce as cost effective a utilization of expensive communication resources as a more tailored architecture would. The headers of Internet packets are fairly long (a typical header is 40 bytes), and if short packets are sent, this overhead is apparent. The worse case, of course, is the single character remote login packets, which carry 40 bytes of header and one byte of data. Actually, it is very difficult for any protocol suite to claim that these sorts of interchanges are carried out with reasonable efficiency. At the other extreme, large packets for file transfer, with perhaps 1,000 bytes of data, have an overhead for the header of only four percent.

Another possible source of inefficiency is retransmission of lost packets. Since Internet does not insist that lost packets be recovered at the network level, it may be necessary to retransmit a lost packet from one end of the Internet to the other. This means that the retransmitted packet may cross several intervening nets a second time, whereas recovery at the network level would not generate this repeat traffic. This is an example of the tradeoff resulting from the decision, discussed above, of providing services from the end-points. The network interface code is much simpler, but the overall efficiency is potentially less. However, if the retransmission rate is low enough (for example, 1%) then the incremental cost is tolerable. As a rough rule of thumb for networks incorporated into the architecture, a loss of one packet in a hundred is quite reasonable, but a loss of one packet in ten suggests that reliability enhancements be added to the network if that type of service is required.

The cost of attaching a host to the Internet is perhaps somewhat higher than in other architectures, because all of the mechanisms to provide the desired types of service, such as acknowledgments and retransmission strategies, must be implemented in the host rather than in the network. Initially, to programmers who were not familiar with protocol implementation, the effort of doing this seemed somewhat daunting. Implementors tried such things as moving the transport protocols to a front end processor, with the idea that the protocols would be implemented only once, rather than again for every type of host. However, this required the invention of a host to front end protocol which some thought almost as complicated to implement as the original transport protocol. As experience with protocols increases, the anxieties associated with implementing a protocol suite within the host seem to be decreasing, and implementations are now available for a wide variety of machines, including personal computers and other machines with very limited computing resources.

A related problem arising from the use of host-resident mechanisms is that poor implementation of the mechanism may hurt the network as well as the host. This problem was tolerated, because the initial experiments involved a limited number of host implementations which could be controlled. However, as the use of Internet has grown, this problem has occasionally surfaced in a serious way. In this respect, the goal of robustness, which led to the method of fate-sharing, which led to host-resident algorithms, contributes to a loss of robustness if the host mis-behaves.

The last goal was accountability. In fact, accounting was discussed in the first paper by Cerf and Kahn as an important function of the protocols and gateways. However, at the present time, the Internet architecture contains few tools for accounting for packet flows. This problem is only now being studied, as the scope of the architecture is being expanded to include non-military consumers who are seriously concerned with understanding and monitoring the usage of the resources within the internet.

8. Architecture and Implementation

The previous discussion clearly suggests that one of the goals of the Internet architecture was to provide wide flexibility in the service offered. Different transport protocols could be used to provide different types of service, and different networks could be incorporated. Put another way, the architecture tried very hard not to constrain the range of service which the Internet could be engineered to provide. This, in turn, means that to understand the service which can be offered by a particular implementation of an Internet, one must look not to the architecture, but to the actual engineering of the software within the particular hosts and gateways, and to the particular networks which have been incorporated. I will use the term "realization" to describe a particular set of networks, gateways and hosts which have been connected together in the context of the Internet architecture. Realizations can differ by orders of magnitude in the service which they offer. Realizations have been built out of 1200 bit per second phone lines, and out of networks only with speeds greater than 1 megabit per second. Clearly, the throughput expectations which one can have of these realizations differ by orders of magnitude. Similarly, some Internet realizations have delays measured in tens of milliseconds, where others have delays measured in seconds. Certain applications such as real time speech work fundamentally differently across these two realizations. Some Internets have been engineered so that there is great redundancy in the gateways and paths. These Internets are survivable, because resources exist which can be reconfigured after failure. Other Internet realizations, to reduce cost, have single points of connectivity through the realization, so that a failure may partition the Internet into two halves.

The Internet architecture tolerates this variety of realization by design. However, it leaves the designer of a particular realization with a great deal of engineering to do. One of the major struggles of this architectural development was to understand how to give guidance to the designer of a realization, guidance which would relate the engineering of the realization to the types of service which would result. For example, the designer must answer the following sort of question. What sort of bandwidths must be in the underlying networks, if the overall service is to deliver a throughput of a certain rate? Given a certain model of possible failures within this realization, what sorts of redundancy ought to be engineered into the realization?

Most of the known network design aids did not seem helpful in answering these sorts of questions. Protocol verifiers, for example, assist in confirming that protocols meet specifications. However, these tools almost never deal with performance issues, which are essential to the idea of the type of service. Instead, they deal with the much more restricted idea of logical correctness of the protocol with respect to specification. While tools to verify logical correctness are useful, both at the specification and implementation stage, they do not help with the severe problems that often arise related to performance. A typical implementation experience is that even after logical correctness has been demonstrated, design faults are discovered that may cause a performance degradation of an order of magnitude. Exploration of this problem has led to the conclusion that the difficulty usually arises, not in the protocol itself, but in the operating system on which the protocol runs. This being the case, it is difficult to address the problem within the context of the architectural specification. However, we still strongly feel the need to give the implementor guidance. We continue to struggle with this problem today.

The other class of design aid is the simulator, which takes a particular realization and explores the service which it can deliver under a variety of loadings. No one has yet attempted to construct a simulator which take into account the wide variability of the gateway implementation, the host implementation, and the network performance which one sees within possible Internet realizations. It is thus the case that the analysis of most Internet realizations is done on the back of an envelope. It is a comment on the goal structure of the Internet architecture that a back of the envelope analysis, if done by a sufficiently knowledgeable person, is usually sufficient. The designer of a particular Internet realization is usually less concerned with obtaining the last five percent possible in line utilization than knowing whether the desired type of service can be achieved at all given the resources at hand at the moment.

The relationship between architecture and performance is an extremely challenging one. The designers of the Internet architecture felt very strongly that it was a serious mistake to attend only to logical correctness and ignore the issue of performance. However, they experienced great difficulty in formalizing any aspect of performance constraint within the architecture. These difficulties arose both because the goal of the architecture was not to constrain performance, but to permit variability, and secondly (and perhaps more fundamentally), because there seemed to be no useful formal tools for describing performance.

This problem was particularly aggravating because the goal of the Internet project was to produce specification documents which were to become military standards. It is a well known problem with government contracting that one cannot expect a contractor to meet any criteria which is not a part of the procurement standard. If the Internet is concerned about performance, therefore, it was mandatory that performance requirements be put into the procurement specification. It was trivial to invent specifications which constrained the performance, for example to specify that the implementation must be capable of passing 1,000 packets a second. However, this sort of constraint could not be part of the architecture, and it was therefore up to the individual performing the procurement to recognize that these performance constraints must be added to the specification, and to specify them properly to achieve a realization which provides the required types of service. We do not have a good idea how to offer guidance in the architecture for the person performing this task.

9. Datagrams

The fundamental architectural feature of the Internet is the use of datagrams as the entity which is transported across the underlying networks. As this paper has suggested, there are several reasons why datagrams are important within the architecture. First, they eliminate the need for connection state within the intermediate switching nodes, which means that the Internet can be reconstituted after a failure without concern about state. Secondly, the datagram provides a basic building block out of which a variety of types of service can be implemented. In contrast to the virtual circuit, which usually implies a fixed type of service, the datagram provides a more elemental service which the endpoints can combine as appropriate to build the type of service needed. Third, the datagram represents the minimum network service assumption, which has permitted a wide variety of networks to be incorporated into various Internet realizations. The decision to use the datagram was an extremely successful one, which allowed the Internet to meet its most important goals very successfully.

There is a mistaken assumption often associated with datagrams, which is that the motivation for datagrams is the support of a higher level service which is essentially equivalent to the datagram. In other words, it has sometimes been suggested that the datagram is provided because the transport service which the application requires is a datagram service. In fact, this is seldom the case. While some applications in the Internet, such as simple queries of date servers or name servers, use an access method based on an unreliable datagram, most services within the Internet would like a more sophisticated transport model than simple datagram. Some services would like the reliability enhanced, some would like the delay smoothed and buffered, but almost all have some expectation more complex than a datagram. It is important to understand that the role of the datagram in this respect is as a building block, and not as a service in itself.

10. TCP

There were several interesting and controversial design decisions in the development of TCP, and TCP itself went through several major versions before it became a reasonably stable standard. Some of these design decisions, such as window management and the nature of the port address structure, are discussed in a series of implementation notes published as part of the TCP protocol handbook.[17, 18] But again the motivation for the decision is sometimes lacking. In this section, I attempt to capture some of the early reasoning that went into parts of TCP. This section is of necessity incomplete; a complete review of the history of TCP itself would require another paper of this length.

The original ARPANET host-to host protocol provided flow control based on both bytes and packets. This seemed overly complex, and the designers of TCP felt that only one form of regulation would be sufficient. The choice was to regulate the delivery of bytes, rather than packets. Flow control and acknowledgment in TCP is thus based on byte number rather than packet number. Indeed, in TCP there is no significance to the packetization of the data.

This decision was motivated by several considerations, some of which became irrelevant and others of which were more important that anticipated. One reason to acknowledge bytes was to permit the insertion of control information into the sequence space of the bytes, so that control as well as data could be acknowledged. That use of the sequence space was dropped, in favor of ad hoc techniques for dealing with each control message. While the original idea has appealing generality, it caused complexity in practice.

A second reason for the byte stream was to permit the TCP packet to be broken up into smaller packets if necessary in order to fit through a net with a small packet size. But this function was moved to the IP layer when IP was split from TCP, and IP was forced to invent a different method of fragmentation.

A third reason for acknowledging bytes rather than packets was to permit a number of small packets to be gathered together into one larger packet in the sending host if retransmission of the data was necessary. It was not clear if this advantage would be important; it turned out to be critical. Systems such as UNIX which have a internal communication model based on single character interactions often send many packets with one byte of data in them. (One might argue from a network perspective that this behavior is silly, but it was a reality, and a necessity for interactive remote login.) It was often observed that such a host could produce a flood of packets with one byte of data, which would arrive much faster than a slow host could process them. The result is lost packets and retransmission.

If the retransmission was of the original packets, the same problem would repeat on every retransmission, with a performance impact so intolerable as to prevent operation. But since the bytes were gathered into one packet for retransmission, the retransmission occurred in a much more effective way which permitted practical operation.

On the other hand, the acknowledgment of bytes could be seen as creating this problem in the first place. If the basis of flow control had been packets rather than bytes, then this flood might never have occurred. Control at the packet level has the effect, however, of providing a severe limit on the throughput if small packets are sent. If the receiving host specifies a number of packets to receive, without any knowledge of the number of bytes in each, the actual amount of data received could vary by a factor of 1000, depending on whether the sending host puts one or one thousand bytes in each packet.

In retrospect, the correct design decision may have been that if TCP is to provide effective support of a variety of services, both packets and bytes must be regulated, as was done in the original ARPANET protocols.

Another design decision related to the byte stream was the End-Of-Letter flag, or EOL. This has now vanished from the protocol, replaced by the Push flag, or PSH. The original idea of EOL was to break the byte stream into records. It was implemented by putting data from separate records into separate packets, which was not compatible with the idea of combining packets on retransmission. So the semantics of EOL was changed to a weaker form, meaning only that the data up to this point in the stream was one or more complete application-level elements, which should occasion a flush of any internal buffering in TCP or the network. By saying "one or more" rather than "exactly one", it became possible to combine several together and preserve the goal of compacting data in reassembly. But the weaker semantics meant that various applications had to invent an ad hoc mechanism for delimiting records on top of the data stream.

In this evolution of EOL semantics, there was a little known intermediate form, which generated great debate. Depending on the buffering strategy of the host, the byte stream model of TCP can cause great problems in one improbable case. Consider a host in which the incoming data is put in a sequence of fixed size buffers. A buffer is returned to the user either when it is full, or an EOL is received. Now consider the case of the arrival of an out-of-order packet which is so far out of order to lie beyond the current buffer. Now further consider that after receiving this out-of-order packet, a packet with an EOL causes the current buffer to be returned to the user only partially full. This particular sequence of actions has the effect of causing the out of order data in the next buffer to be in the wrong place, because of the empty bytes in the buffer returned to the user. Coping with this generated book-keeping problems in the host which seemed unnecessary.

To cope with this it was proposed that the EOL should "use up" all the sequence space up to the next value which was zero mod the buffer size. In other words, it was proposed that EOL should be a tool for mapping the byte stream to the buffer management of the host. This idea was not well received at the time, as it seemed much too ad hoc, and only one host seemed to have this problem.[2] In retrospect, it may have been the correct idea to incorporate into TCP some means of relating the sequence space and the buffer management algorithm of the host. At the time, the designers simply lacked the insight to see how that might be done in a sufficiently general manner.

[2]This use of EOL was properly called "Rubber EOL" but its detractors quickly called it "rubber baby buffer bumpers" in an attempt to ridicule the idea. Credit must go to the creator of the idea, Bill Plummer, for sticking to his guns in the face of detractors saying the above to him ten times fast.

11. Conclusion

In the context of its priorities, the Internet architecture has been very successful. The protocols are widely used in the commercial and military environment, and have spawned a number of similar architectures. At the same time, its success has made clear that in certain situations, the priorities of the designers do not match the needs of the actual users. More attention to such things as accounting, resource management and operation of regions with separate administrations are needed.

While the datagram has served very well in solving the most important goals of the Internet, it has not served so well when we attempt to address some of the goals which were further down the priority list. For example, the goals of resource management and accountability have proved difficult to achieve in the context of datagrams. As the previous section discussed, most datagrams are a part of some sequence of packets from source to destination, rather than isolated units at the application level. However, the gateway cannot directly see the existence of this sequence, because it is forced to deal with each packet in isolation. Therefore, resource management decisions or accounting must be done on each packet separately. Imposing the datagram model on the internet layer has deprived that layer of an important source of information which it could use in achieving these goals.

This suggests that there may be a better building block than the datagram for the next generation of architecture. The general characteristic of this building block is that it would identify a sequence of packets traveling from the source to the destination, without assuming any particular type of service with that service. I have used the word "flow" to characterize this building block. It would be necessary for the gateways to have flow state in order to remember the nature of the flows which are passing through them, but the state information would not be critical in maintaining the desired type of service associated with the flow. Instead, that type of service would be enforced by the end points, which would periodically send messages to ensure that the proper type of service was being associated with the flow. In this way, the state information associated with the flow could be lost in a crash without permanent disruption of the service features being used. I call this concept "soft state," and it may very well permit us to achieve our primary goals of survivability and flexibility, while at the same time doing a better job of dealing with the issue of resource management and accountability. Exploration of alternative building blocks constitute one of the current directions for research within the DARPA Internet Program.

flow ID available in IPv6

12. Acknowledgments -- A Historical Perspective

It would be impossible to acknowledge all the contributors to the Internet project; there have literally been hundreds over the 15 years of development: designers, implementors, writers and critics. Indeed, an important topic, which probably deserves a paper in itself, is the process by which this project was managed. The participants came from universities, research laboratories and corporations, and they united (to some extent) to achieve this common goal.

The original vision for TCP came from Robert Kahn and Vinton Cerf, who saw very clearly, back in 1973, how a protocol with suitable features might be the glue that would pull together the various emerging network technologies. From their position at DARPA, they guided the project in its early days to the point where TCP and IP became standards for the DoD.

The author of this paper joined the project in the mid-70s, and took over architectural responsibility for TCP/IP in 1981. He would like to thank all those who have worked with him, and particularly those who took the time to reconstruct some of the lost history in this paper.

References

1. V. Cerf, and R. Kahn, "A Protocol for Packet Network Intercommunication", *IEEE Transactions on Communications*, Vol. Com-22, No. 5, May 1974, pp. 637-648.

2. ISO, "Transport Protocol Specification", Tech. report IS-8073, International Organization for Standardization, September 1984.

3. ISO, "Protocol for Providing the Connectionless-Mode Network Service", Tech. report DIS8473, International Organization for Standardization, 1986.

4. R. Callon, "Internetwork Protocol", *Proceedings of the IEEE*, Vol. 71, No. 12, December 1983, pp. 1388-1392.

5. Jonathan B. Postel, "Internetwork Protocol Approaches", *IEEE Transactions on Communications*, Vol. Com-28, No. 4, April 1980, pp. 605-611.

6. Jonathan B. Postel, Carl A. Sunshine, Danny Cohen, "The ARPA Internet Protocol", *Computer Networks 5*, Vol. 5, No. 4, July 1981, pp. 261-271.

7. Alan Sheltzer, Robert Hinden, and Mike Brescia, "Connecting Different Types of Networks with Gateways", *Data Communications*, August 1982.

8. J. McQuillan and D. Walden, "The ARPA Network Design Decisions", *Computer Networks*, Vol. 1, No. 5, August 1977, pp. 243-289.

9. R.E. Kahn, S.A. Gronemeyer, J. Burdifiel, E.V. Hoversten, "Advances in Packet Radio Technology", *Proceedings of the IEEE*, Vol. 66, No. 11, November 1978, pp. 1408-1496.

10. B.M. Leiner, D.L. Nelson, F.A. Tobagi, "Issues in Packet Radio Design", *Proceedings of the IEEE*, Vol. 75, No. 1, January 1987, pp. 6-20.

11. -, "Transmission Control Protocol RFC-793", *DDN Protocol Handbook*, Vol. 2, September 1981, pp. 2.179-2.198.

12. Jack Haverty, "XNET Formats for Internet Protocol Version 4 IEN 158", *DDN Protocol Handbook*, Vol. 2, October 1980, pp. 2-345 to 2-348.

13. Jonathan Postel, "User Datagram Protocol NIC-RFC-768", *DDN Protocol Handbook*, Vol. 2. August 1980, pp. 2.175-2.177.

14. I. Jacobs, R. Binder, and E. Hoversten, "General Purpose Packet Satellite Networks", *Proceedings of the IEEE*, Vol. 66, No. 11, November 1978, pp. 1448-1467.

15. C. Topolcic and J. Kaiser, "The SATNET Monitoring System", *Proceedings of the IEEE-MILCOM Boston, MA*, October 1985, pp. 26.1.1-26.1.9.

16. W.Edmond, S.Blumenthal, A.Echenique, S.Storch, T.Calderwood, and T.Rees, "The Butterfly Satellite IMP for the Wideband Packet Satellite Network", *Proceedings of the ACM SIGCOMM '86*, ACM, Stowe, Vt., August 1986, pp. 194-203.

17. David D. Clark, "Window and Acknowledgment Strategy in TCP NIC-RFC-813", *DDN Protocol Handbook*, Vol. 3, July 1982, pp. 3-5 to 3-26.

18. David D. Clark, "Name, Addresses, Ports, and Routes NIC-RFC-814", *DDN Protocol Handbook*, Vol. 3, July 1982, pp. 3-27 to 3-40.