A Protocol for Packet Network Intercommunication

VINTON G. CERF AND ROBERT E. KAHN, MEMBER, IEEE

Abstract—**A protocol that supports the sharing of resources that exist in different packet switching networks is presented. The protocol provides for variation in individual network packet sizes, transmission failures, sequencing, flow control, end-to-end error checking, and the creation and destruction of logical process-to-process connections. Some implementation issues are considered, and problems such as internetwork routing, accounting, and timeouts are exposed.**

INTRODUCTION

IN THE LAST few years considerable effort has been expended on the design and implementation of packet switching networks [1]–[7],[14],[17]. A principle reason for developing such networks has been to facilitate the sharing of computer resources. A packet communication network includes a transportation mechanism for delivering data between computers or between computers and terminals. To make the data meaningful, computers and terminals share a common protocol (i.e., a set of agreed upon conventions). Several protocols have already been developed for this purpose [8]–[12],[16]. However, these protocols have addressed only the problem of communication on the same network. In this paper we present a protocol design and philosophy that supports the sharing of resources that exist in different packet switching networks.

After a brief introduction to internetwork protocol issues, we describe the function of a GATEWAY as an interface between networks and discuss its role in the protocol. We then consider the various details of the protocol, including addressing, formatting, buffering, sequencing, flow control, error control, and so forth. We close with a description of an interprocess communication mechanism and show how it can be supported by the internetwork protocol.

Even though many different and complex problems must be solved in the design of an individual packet switching network, these problems are manifestly compounded when dissimilar networks are interconnected. Issues arise which may have no direct counterpart in an individual network and which strongly influence the way in which internetwork communication can take place.

A typical packet switching network is composed of a set of computer resources called HOSTS, a set of one or more *packet switches*, and a collection of communication media that interconnect the packet switches. Within each HOST, we assume that there exist *processes* which must communicate with processes in their own or other HOSTS. Any current definition of a process will be adequate for our purposes [13]. These processes are generally the ultimate source and destination of data in the network. Typically, within an individual network, there exists a protocol for communication between any source and destination process. Only the source and destination processes require knowledge of this convention for communication to take place. Processes in two distinct networks would ordinarily use different protocols for this purpose. The ensemble of packet switches and communication media is called the *packet switching subnet*. Fig. 1 illustrates these ideas.

In a typical packet switching subnet, data of a fixed maximum size are accepted from a source HOST, together with a formatted destination address which is used to route the data in a store and forward fashion. The transmit time for this data is usually dependent upon internal network parameters such as communication media data rates, buffering and signaling strategies, routing, propagation delays, etc. In addition, some mechanism is generally present for error handling and determination of status of the networks components.

Individual packet switching networks may differ in their implementations as follows.

1) Each network may have distinct ways of addressing the receiver, thus requiring that a uniform addressing scheme be created which can be understood by each individual network.

2) Each network may accept data of different maximum size, thus requiring networks to deal in units of the smallest maximum size (which may be impractically small) or requiring procedures which allow data crossing a network boundary to be reformatted into smaller pieces.

3) The success or failure of a transmission and its performance in each network is governed by different time delays in accepting, delivering, and transporting the data. This requires careful development of internetwork timing procedures to insure that data can be successfully delivered through the various networks.

4) Within each network, communication may be disrupted due to unrecoverable mutation of the data or missing data. End-to-end restoration procedures are desirable to allow complete recovery from these conditions.

Paper approved by the Associate Editor for Data Communications of the IEEE Communications Society for publication without oral presentation. Manuscript received November 5, 1973. The research reported in this paper was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC 15-73-C-0370.

V. G. Cerf is with the Department of Computer Science and Electrical Engineering, Stanford University, Stanford, Calif.

R. E. Kahn is with the Information Processing Technology Office, Advanced Research Projects Agency, Department of Defense, Arlington, Va.

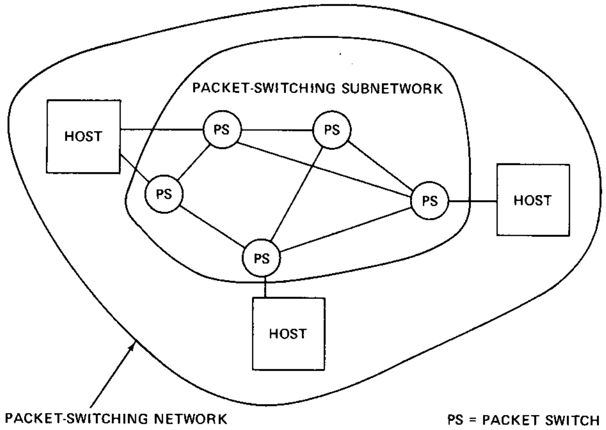

Fig. 1. Typical packet switching network.

5) Status information, routing, fault detection, and isolation are typically different in each network. Thus, to obtain verification of certain conditions, such as an inaccessible or dead destination, various kinds of coordination must be invoked between the communicating networks.

It would be extremely convenient if all the differences between networks could be economically resolved by suitable interfacing at the network boundaries. For many of the differences, this objective can be achieved. However, both economic and technical considerations lead us to prefer that the interface be as simple and reliable as possible and deal primarily with passing data between networks that use different packet switching strategies.

The question now arises as to whether the interface ought to account for differences in HOST or process level protocols by transforming the source conventions into the corresponding destination conventions. We obviously want to allow conversion between packet switching strategies at the interface, to permit interconnection of existing and planned networks. However, the complexity and dissimilarity of the HOST or process level protocols makes it desirable to avoid having to transform between them at the interface, even if this transformation were always possible. Rather, compatible HOST and process level protocols must be developed to achieve effective internetwork resource sharing. The unacceptable alternative is for every HOST or process to implement every protocol (a potentially unbounded number) that may be needed to communicate with other networks. We therefore assume that a common protocol is to be used between HOST's or processes in different networks and that the interface between networks should take as small a role as possible in this protocol.

To allow networks under different ownership to interconnect, some accounting will undoubtedly be needed for traffic that passes across the interface. In its simplest terms, this involves an accounting of packets handled by each net for which charges are passed from net to net until the buck finally stops at the user or his representative. Furthermore, the interconnection must preserve intact the internal operation of each individual network. This is easily achieved if two networks interconnect as if each were a HOST to the other network, but without utilizing or indeed incorporating any elaborate HOST protocol transformations.

It is thus apparent that the interface between networks must play a central role in the development of any network interconnection strategy. We give a special name to this interface that performs these functions and call it a GATEWAY.

THE GATEWAY NOTION

In Fig. 2 we illustrate three individual networks labeled A, B, and C which are joined by GATEWAYS M and N. GATEWAY M interfaces network A with network B, and GATEWAY N interfaces network B to network C. We assume that an individual network may have more than one GATEWAY (e.g., network B) and that there may be more than one GATEWAY path to use in going between a pair of networks. The responsibility for properly routing data resides in the GATEWAY.

In practice, a GATEWAY between two networks may be composed of two halves, each associated with its own network. It is possible to implement each half of a GATEWAY so it need only embed internetwork packets in local packet format or extract them. We propose that the GATEWAYS handle internetwork packets in a standard format, but we are not proposing any particular transmission procedure between GATEWAY halves.

Let us now trace the flow of data through the interconnected networks. We assume a packet of data from process X enters network A destined for process Y in network C. The address of Y is initially specified by process X and the address of GATEWAY M is derived from the address of process Y. We make no attempt to specify whether the choice of GATEWAY is made by process X, its HOST, or one of the packet switches in network A. The packet traverses network A until it reaches GATEWAY M. At the GATEWAY, the packet is reformatted to meet the requirements of network B, account is taken of this unit of flow between A and B, and the GATEWAY delivers the packet to network B. Again the derivation of the next GATEWAY address is accomplished based on the address of the destination Y. In this case, GATEWAY N is the next one. The packet traverses network B until it finally reaches GATEWAY N where it is formatted to meet the requirements of network C. Account is again taken of this unit of flow between networks B and C. Upon entering network C, the packet is routed to the HOST in which process Y resides and there it is delivered to its ultimate destination.

Since the GATEWAY must understand the address of the source and destination HOSTS, this information must be available in a standard format in every packet which arrives at the GATEWAY. This information is contained in an *internetwork header* prefixed to the packet by the source HOST. The packet format, including the internet-

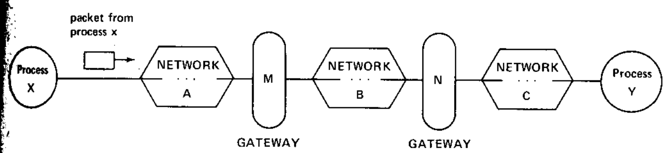

Fig. 2. Three networks interconnected by two GATEWAYS.

(may be null)	← Internetwork Header				→		
LOCAL HEADER	SOURCE	DESTINATION	SEQUENCE NO.	BYTE COUNT	FLAG FIELD	TEXT	CHECKSUM

Fig. 3. Internetwork packet format (fields not shown to scale).

work header, is illustrated in Fig. 3. The source and destination entries uniformly and uniquely identify the address of every HOST in the composite network. Addressing is a subject of considerable complexity which is discussed in greater detail in the next section. The next two entries in the header provide a sequence number and a byte count that may be used to properly sequence the packets upon delivery to the destination and may also enable the GATEWAYS to detect fault conditions affecting the packet. The flag field is used to convey specific control information and is discussed in the section on retransmission and duplicate detection later. The remainder of the packet consists of text for delivery to the destination and a trailing check sum used for end-to-end software verification. The GATEWAY does *not* modify the text and merely forwards the check sum along without computing or recomputing it.

Each network may need to augment the packet format before it can pass through the individual network. We have indicated a *local header* in the figure which is prefixed to the beginning of the packet. This local header is introduced merely to illustrate the concept of embedding an internetwork packet in the format of the individual network through which the packet must pass. It will obviously vary in its exact form from network to network and may even be unnecessary in some cases. Although not explicitly indicated in the figure, it is also possible that a local trailer may be appended to the end of the packet.

Unless all transmitted packets are legislatively restricted to be small enough to be accepted by every individual network, the GATEWAY may be forced to split a packet into two or more smaller packets. This action is called fragmentation and must be done in such a way that the destination is able to piece together the fragmented packet. It is clear that the internetwork header format imposes a minimum packet size which all networks must carry (obviously all networks will want to carry packets larger than this minimum). We believe the long range growth and development of internetwork communication would be seriously inhibited by specifying how much larger than the minimum a packet size can be, for the following reasons.

1) If a maximum permitted packet size is specified then it becomes impossible to completely isolate the internal packet size parameters of one network from the internal packet size parameters of all other networks.

2) It would be very difficult to increase the maximum permitted packet size in response to new technology (e.g., large memory systems, higher data rate communication facilities, etc.) since this would require the agreement and then implementation by all participating networks.

3) Associative addressing and packet encryption may require the size of a particular packet to expand during transit for incorporation of new information.

Provision for fragmentation (regardless of where it is performed) permits packet size variations to be handled on an individual network basis without global administration and also permits HOSTS and processes to be insulated from changes in the packet sizes permitted in any networks through which their data must pass.

If fragmentation must be done, it appears best to do it upon entering the next network at the GATEWAY since only this GATEWAY (and not the other networks) must be aware of the internal packet size parameters which made the fragmentation necessary.

If a GATEWAY fragments an incoming packet into two or more packets, they must eventually be passed along to the destination HOST as fragments or reassembled for the HOST. It is conceivable that one might desire the GATEWAY to perform the reassembly to simplify the task of the destination HOST (or process) and/or to take advantage of a larger packet size. We take the position that GATEWAYS should not perform this function since GATEWAY reassembly can lead to serious buffering problems, potential deadlocks, the necessity for all fragments of a packet to pass through the same GATEWAY, and increased delay in transmission. Furthermore, it is not sufficient for the GATEWAYS to provide this function since the final GATEWAY may also have to fragment a packet for transmission. Thus the destination HOST must be prepared to do this task.

Let us now turn briefly to the somewhat unusual accounting effect which arises when a packet may be fragmented by one or more GATEWAYS. We assume, for simplicity, that each network initially charges a fixed rate per packet transmitted, regardless of distance, and if one network can handle a larger packet size than another, it charges a proportionally larger price per packet. We also assume that a subsequent increase in any network's packet size does not result in additional cost per packet to its users. The charge to a user thus remains basically constant through any net which must fragment a packet. The unusual effect occurs when a packet is fragmented into smaller packets which must individually pass through a subsequent network with a larger packet size than the original unfragmented packet. We expect that most networks will naturally select packet sizes close to one another, but in any case, an increase in packet size in one net, even when it causes fragmentation, will not increase the cost of transmission and may actually decrease it. In the event that any other packet charging policies (than the one we suggest) are adopted, differences in cost can be used as an economic lever toward optimization of individual network performance.

PROCESS LEVEL COMMUNICATION

We suppose that processes wish to communicate in full duplex with their correspondents using unbounded but finite length messages. A single character might constitute the text of a message from a process to a terminal or vice versa. An entire page of characters might constitute the text of a message from a file to a process. A data stream (e.g., a continuously generated bit string) can be represented as a sequence of finite length messages.

Within a HOST we assume the existence of a transmission control program (TCP) which handles the transmission and acceptance of messages on behalf of the processes it serves. The TCP is in turn served by one or more packet switches connected to the HOST in which the TCP resides. Processes that want to communicate present messages to the TCP for transmission, and TCP's deliver incoming messages to the appropriate destination processes. We allow the TCP to break up messages into segments because the destination may restrict the amount of data that may arrive, because the local network may limit the maximum transmission size, or because the TCP may need to share its resources among many processes concurrently. Furthermore, we constrain the length of a segment to an integral number of 8-bit bytes. This uniformity is most helpful in simplifying the software needed with HOST machines of different natural word lengths. Provision at the process level can be made for padding a message that is not an integral number of bytes and for identifying which of the arriving bytes of text contain information of interest to the receiving process.

Multiplexing and demultiplexing of segments among processes are fundamental tasks of the TCP. On transmission, a TCP must multiplex together segments from different source processes and produce internetwork packets for delivery to one of its serving packet switches. On reception, a TCP will accept a sequence of packets from its serving packet switch(es). From this sequence of arriving packets (generally from different HOSTS), the TCP must be able to reconstruct and deliver messages to the proper destination processes.

We assume that every segment is augmented with additional information that allows transmitting and receiving TCP's to identify destination and source processes, respectively. At this point, we must face a major issue. How should the source TCP format segments destined for the same destination TCP? We consider two cases.

Case 1): If we take the position that segment boundaries are immaterial and that a byte stream can be formed of segments destined for the same TCP, then we may gain improved transmission efficiency and resource sharing by arbitrarily parceling the stream into packets, permitting many segments to share a single internetwork packet header. However, this position results in the need to reconstruct exactly, and in order, the stream of text bytes produced by the source TCP. At the destination, this stream must first be parsed into segments and these in turn must be used to reconstruct messages for delivery to the appropriate processes.

There are fundamental problems associated with this strategy due to the possible arrival of packets out of order at the destination. The most critical problem appears to be the amount of interference that processes sharing the same TCP–TCP byte stream may cause among themselves. This is especially so at the receiving end. First, the TCP may be put to some trouble to parse the stream back into segments and then distribute them to buffers where messages are reassembled. If it is not readily apparent that all of a segment has arrived (remember, it may come as several packets), the receiving TCP may have to suspend parsing temporarily until more packets have arrived. Second, if a packet is missing, it may not be clear whether succeeding segments, even if they are identifiable, can be passed on to the receiving process, unless the TCP has knowledge of some process level sequencing scheme. Such knowledge would permit the TCP to decide whether a succeeding segment could be delivered to its waiting process. Finding the beginning of a segment when there are gaps in the byte stream may also be hard.

Case 2): Alternatively, we might take the position that the destination TCP should be able to determine, upon its arrival and without additional information, for which process or processes a received packet is intended, and if so, whether it should be delivered then.

If the TCP is to determine for which process an arriving packet is intended, every packet must contain a *process header* (distinct from the internetwork header) that completely identifies the destination process. For simplicity, we assume that each packet contains text from a single process which is destined for a single process. Thus each packet need contain only one process header. To decide whether the arriving data is deliverable to the destination process, the TCP must be able to determine whether the data is in the proper sequence (we can make provision for the destination process to instruct its TCP to ignore sequencing, but this is considered a special case). With the assumption that each arriving packet contains a process header, the necessary sequencing and destination process identification is immediately available to the destination TCP.

Both Cases 1) and 2) provide for the demultiplexing and delivery of segments to destination processes, but only Case 2) does so without the introduction of potential interprocess interference. Furthermore, Case 1) introduces extra machinery to handle flow control on a HOST-to-HOST basis, since there must also be some provision for process level control, and this machinery is little used since the probability is small that within a given HOST, two processes will be coincidentally scheduled to send messages to the same destination HOST. For this reason, we select the method of Case 2) as a part of the *internetwork transmission protocol.*

ADDRESS FORMATS

The selection of address formats is a problem between networks because the local network addresses of TCP's may vary substantially in format and size. A uniform internetwork TCP address space, understood by each GATEWAY and TCP, is essential to routing and delivery of internetwork packets.

Similar troubles are encountered when we deal with process addressing and, more generally, port addressing. We introduce the notion of *ports* in order to permit a process to distinguish between multiple message streams. The port is simply a designator of one such message stream associated with a process. The means for identifying a port are generally different in different operating systems, and therefore, to obtain uniform addressing, a standard port address format is also required. A port address designates a full duplex message stream.

TCP ADDRESSING

TCP addressing is intimately bound up in routing issues, since a HOST or GATEWAY must choose a suitable destination HOST or GATEWAY for an outgoing internetwork packet. Let us postulate the following address format for the TCP address (Fig. 4). The choice for network identification (8 bits) allows up to 256 distinct networks. This size seems sufficient for the forseeable future. Similarly, the TCP identifier field permits up to 65 536 distinct TCP's to be addressed, which seems more than sufficient for any given network.

As each packet passes through a GATEWAY, the GATEWAY observes the destination network ID to determine how to route the packet. If the destination network is connected to the GATEWAY, the lower 16 bits of the TCP address are used to produce a local TCP address in the destination network. If the destination network is not connected to the GATEWAY, the upper 8 bits are used to select a subsequent GATEWAY. We make no effort to specify how each individual network shall associate the internetwork TCP identifier with its local TCP address. We also do not rule out the possibility that the local network understands the internetwork addressing scheme and thus alleviates the GATEWAY of the routing responsibility.

PORT ADDRESSING

A receiving TCP is faced with the task of demultiplexing the stream of internetwork packets it receives and reconstructing the original messages for each destination process. Each operating system has its own internal means of identifying processes and ports. We assume that 16 bits are sufficient to serve as internetwork port identifiers. A sending process need not know how the destination port identification will be used. The destination TCP will be able to parse this number appropriately to find the proper buffer into which it will place arriving packets. We permit a large port number field to support processes which want to distinguish between many different messages streams concurrently. In reality, we do not care how the 16 bits are sliced up by the TCP's involved.

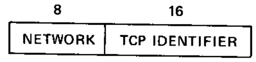

Fig. 4. TCP address.

Even though the transmitted port name field is large, it is still a compact external name for the internal representation of the port. The use of short names for port identifiers is often desirable to reduce transmission overhead and possibly reduce packet processing time at the destination TCP. Assigning short names to each port, however, requires an initial negotiation between source and destination to agree on a suitable short name assignment, the subsequent maintenance of conversion tables at both the source and the destination, and a final transaction to release the short name. For dynamic assignment of port names, this negotiation is generally necessary in any case.

SEGMENT AND PACKET FORMATS

As shown in Fig. 5, messages are broken by the TCP into segments whose format is shown in more detail in Fig. 6. The field lengths illustrated are merely suggestive. The first two fields (source port and destination port in the figure) have already been discussed in the preceding section on addressing. The uses of the third and fourth fields (window and acknowledgment in the figure) will be discussed later in the section on retransmission and duplicate detection.

We recall from Fig. 3 that an internetwork header contains both a sequence number and a byte count, as well as a flag field and a check sum. The uses of these fields are explained in the following section.

REASSEMBLY AND SEQUENCING

The reconstruction of a message at the receiving TCP clearly requires[1] that each internetwork packet carry a sequence number which is unique to its particular destination port message stream. The sequence numbers must be monotonic increasing (or decreasing) since they are used to reorder and reassemble arriving packets into a message. If the space of sequence numbers were infinite, we could simply assign the next one to each new packet. Clearly, this space cannot be infinite, and we will consider what problems a finite sequence number space will cause when we discuss retransmission and duplicate detection in the next section. We propose the following scheme for performing the sequencing of packets and hence the reconstruction of messages by the destination TCP.

A pair of ports will exchange one or more messages over a period of time. We could view the sequence of messages produced by one port as if it were embedded in an infinitely long stream of bytes. Each byte of the message has a unique sequence number which we take to be its byte location relative to the beginning of the stream. When a

[1] In the case of encrypted packets, a preliminary stage of reassembly may be required prior to decryption.

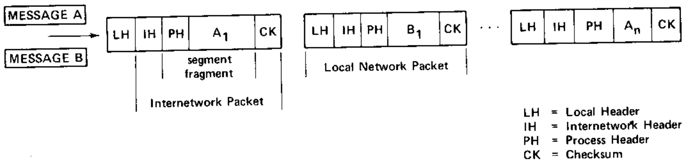

Fig. 5. Creation of segments and packets from messages.

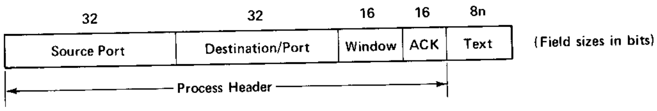

Fig. 6. Segment format (process header and text).

segment is extracted from the message by the source TCP and formatted for internetwork transmission, the relative location of the first byte of segment text is used as the sequence number for the packet. The byte count field in the internetwork header accounts for all the text in the segment (but does not include the check-sum bytes or the bytes in either internetwork or process header). We emphasize that the sequence number associated with a given packet is unique only to the pair of ports that are communicating (see Fig. 7). Arriving packets are examined to determine for which port they are intended. The sequence numbers on each arriving packet are then used to determine the relative location of the packet text in the messages under reconstruction. We note that this allows the exact position of the data in the reconstructed message to be determined even when pieces are still missing.

Every segment produced by a source TCP is packaged in a single internetwork packet and a check sum is computed over the text and process header associated with the segment.

The splitting of messages into segments by the TCP and the potential splitting of segments into smaller pieces by GATEWAYS creates the necessity for indicating to the destination TCP when the end of a segment (ES) has arrived and when the end of a message (EM) has arrived. The flag field of the internetwork header is used for this purpose (see Fig. 8).

The ES flag is set by the source TCP each time it prepares a segment for transmission. If it should happen that the message is completely contained in the segment, then the EM flag would also be set. The EM flag is also set on the last segment of a message, if the message could not be contained in one segment. These two flags are used by the destination TCP, respectively, to discover the presence of a check sum for a given segment and to discover that a complete message has arrived.

The ES and EM flags in the internetwork header are known to the GATEWAY and are of special importance when packets must be split apart for propagation through the next local network. We illustrate their use with an example in Fig. 9.

The original message A in Fig. 9 is shown split into two segments A_1 and A_2 and formatted by the TCP into a pair

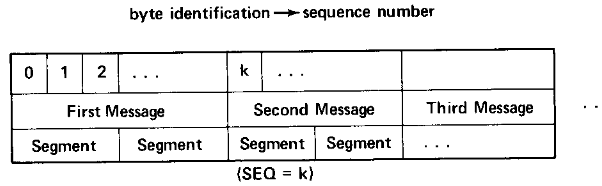

Fig. 7. Assignment of sequence numbers.

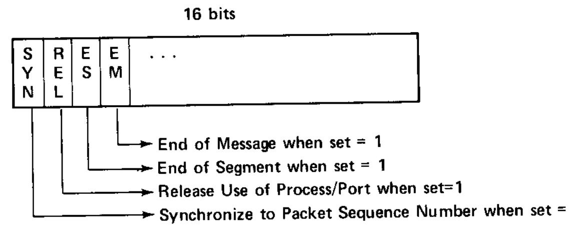

Fig. 8. Internetwork header flag field.

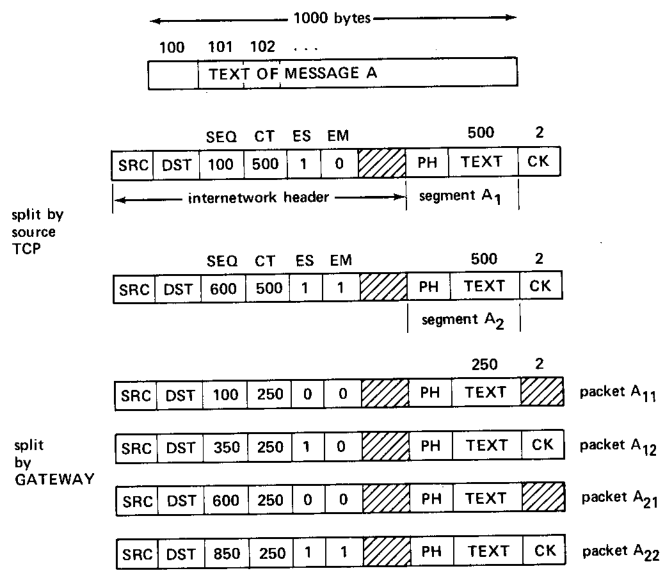

Fig. 9. Message splitting and packet splitting.

of internetwork packets. Packets A_1 and A_2 have their ES bits set, and A_2 has its EM bit set as well. When packet A_1 passes through the GATEWAY, it is split into two pieces: packet A_{11} for which neither EM nor ES bits are set, and packet A_{12} whose ES bit is set. Similarly, packet A_2 is split such that the first piece, packet A_{21}, has neither bit set, but packet A_{22} has both bits set. The sequence number field (SEQ) and the byte count field (CT) of each packet is modified by the GATEWAY to properly identify the text bytes of each packet. The GATEWAY need only examine the internetwork header to do fragmentation.

The destination TCP, upon reassembling segment A_1, will detect the ES flag and will verify the check sum it knows is contained in packet A_{12}. Upon receipt of packet A_{22}, assuming all other packets have arrived, the destination TCP detects that it has reassembled a complete message and can now advise the destination process of its receipt.

RETRANSMISSION AND DUPLICATE DETECTION

No transmission can be 100 percent reliable. We propose a timeout and positive acknowledgment mechanism which will allow TCP's to recover from packet losses from one HOST to another. A TCP transmits packets and waits for replies (acknowledgements) that are carried in the reverse packet stream. If no acknowledgment for a particular packet is received, the TCP will retransmit. It is our expectation that the HOST level retransmission mechanism, which is described in the following paragraphs, will not be called upon very often in practice. Evidence already exists[2] that individual networks can be effectively constructed without this feature. However, the inclusion of a HOST retransmission capability makes it possible to recover from occasional network problems and allows a wide range of HOST protocol strategies to be incorporated. We envision it will occasionally be invoked to allow HOST accommodation to infrequent overdemands for limited buffer resources, and otherwise not used much.

Any retransmission policy requires some means by which the receiver can detect duplicate arrivals. Even if an infinite number of distinct packet sequence numbers were available, the receiver would still have the problem of knowing how long to remember previously received packets in order to detect duplicates. Matters are complicated by the fact that only a finite number of distinct sequence numbers are in fact available, and if they are reused, the receiver must be able to distinguish between new transmissions and retransmissions.

A *window* strategy, similar to that used by the French CYCLADES system (voie virtuelle transmission mode [8]) and the ARPANET very distant HOST connection [18], is proposed here (see Fig. 10).

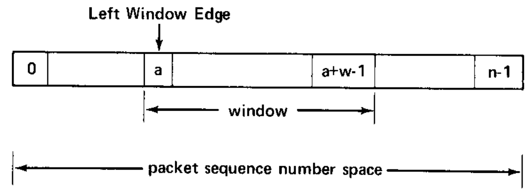

Fig. 10. The window concept.

1	Source Address	
2	Destination Address	
3	Next Packet Seq.	
4	Current Buffer Size	
5	Next Write Position	
6	Next Read Position	
7	End Read Position	
8	No. Retrans.	Max Retrans.
9	Timeout	Flags
10	Curr. Ack	Window

Fig. 11. Conceptual TCB format.

Suppose that the sequence number field in the internetwork header permits sequence numbers to range from 0 to $n - 1$. We assume that the sender will not transmit more than w bytes without receiving an acknowledgment. The w bytes serve as the window (see Fig. 11). Clearly, w must be less than n. The rules for sender and receiver are as follows.

Sender: Let L be the sequence number associated with the left window edge.

1) The sender transmits bytes from segments whose text lies between L and up to $L + w - 1$.

2) On timeout (duration unspecified), the sender retransmits unacknowledged bytes.

3) On receipt of acknowledgment consisting of the receiver's current left window edge, the sender's left window edge is advanced over the acknowledged bytes (advancing the right window edge implicitly).

Receiver:

1) Arriving packets whose sequence numbers coincide with the receiver's current left window edge are acknowledged by sending to the source the next sequence number expected. This effectively acknowledges bytes in between. The left window edge is advanced to the next sequence number expected.

2) Packets arriving with a sequence number to the left of the window edge (or, in fact, outside of the window) are discarded, and the current left window edge is returned as acknowledgment.

3) Packets whose sequence numbers lie within the receiver's window but do not coincide with the receiver's left window edge are optionally kept or discarded, but are not acknowledged. This is the case when packets arrive out of order.

We make some observations on this strategy. First, all computations with sequence numbers and window edges must be made modulo n (e.g., byte 0 follows byte $n - 1$). Second, w must be less than $n/2$[3]; otherwise a retransmission may appear to the receiver to be a new transmission in the case that the receiver has accepted a window's worth of incoming packets, but all acknowledgments have been lost. Third, the receiver can either save or discard arriving packets whose sequence numbers do not coincide with the receiver's left window. Thus, in the simplest implementation, the receiver need not buffer more than one packet per message stream if space is critical. Fourth, multiple packets can be acknowledged simultaneously. Fifth, the receiver is able to deliver messages to processes in their proper order as a natural result of the reassembly mechanism. Sixth, when duplicates are detected, the acknowledgment method used naturally works to resynchronize sender and receiver. Furthermore, if the receiver accepts packets whose sequence numbers lie within the current window but

[2] The ARPANET is one such example.

[3] Actually $n/2$ is merely a convenient number to use; it is only required that a retransmission not appear to be a new transmission.

which are not coincident with the left window edge, an acknowledgment consisting of the current left window edge would act as a stimulus to cause retransmission of the unacknowledged bytes. Finally, we mention an overlap problem which results from retransmission, packet splitting, and alternate routing of packets through different GATEWAYS.

A 600-byte packet might pass through one GATEWAY and be broken into two 300-byte packets. On retransmission, the same packet might be broken into three 200-byte packets going through a different GATEWAY. Since each byte has a sequence number, there is no confusion at the receiving TCP. We leave for later the issue of initially synchronizing the sender and receiver left window edges and the window size.

FLOW CONTROL

Every segment that arrives at the destination TCP is ultimately acknowledged by returning the sequence number of the next segment which must be passed to the process (it may not yet have arrived).

Earlier we described the use of a sequence number space and window to aid in duplicate detection. Acknowledgments are carried in the process header (see Fig. 6) and along with them there is provision for a "suggested window" which the receiver can use to control the flow of data from the sender. This is intended to be the main component of the process flow control mechanism. The receiver is free to vary the window size according to any algorithm it desires so long as the window size never exceeds half the sequence number space.[3]

This flow control mechanism is exceedingly powerful and flexible and does not suffer from synchronization troubles that may be encountered by incremental buffer allocation schemes [9],[10]. However, it relies heavily on an effective retransmission strategy. The receiver can reduce the window even while packets are en route from the sender whose window is presently larger. The net effect of this reduction will be that the receiver may discard incoming packets (they may be outside the window) and reiterate the current window size along with a current window edge as acknowledgment. By the same token, the sender can, upon occasion, choose to send more than a window's worth of data on the possibility that the receiver will expand the window to accept it (of course, the sender must not send more than half the sequence number space at any time). Normally, we would expect the sender to abide by the window limitation. Expansion of the window by the receiver merely allows more data to be accepted. For the receiving HOST with a small amount of buffer space, a strategy of discarding all packets whose sequence numbers do not coincide with the current left edge of the window is probably necessary, but it will incur the expense of extra delay and overhead for retransmission.

TCP INPUT/OUTPUT HANDLING

The TCP has a component which handles input/output (I/O) to and from the network.[4] When a packet has arrived, it validates the addresses and places the packet on a queue. A pool of buffers can be set up to handle arrivals, and if all available buffers are used up, succeeding arrivals can be discarded since unacknowledged packets will be retransmitted.

On output, a smaller amount of buffering is needed, since process buffers can hold the data to be transmitted. Perhaps double buffering will be adequate. We make no attempt to specify how the buffering should be done, except to require that it be able to service the network with as little overhead as possible. Packet sized buffers, one or more ring buffers, or any other combination are possible candidates.

When a packet arrives at the destination TCP, it is placed on a queue which the TCP services frequently. For example, the TCP could be interrupted when a queue placement occurs. The TCP then attempts to place the packet text into the proper place in the appropriate process receive buffer. If the packet terminates a segment, then it can be checksummed and acknowledged. Placement may fail for several reasons.

1) The destination process may not be prepared to receive from the stated source, or the destination port ID may not exist.

2) There may be insufficient buffer space for the text.

3) The beginning sequence number of the text may not coincide with the next sequence number to be delivered to the process (e.g., the packet has arrived out of order).

In the first case, the TCP should simply discard the packet (thus far, no provision has been made for error acknowledgments). In the second and third cases, the packet sequence number can be inspected to determine whether the packet text lies within the legitimate window for reception. If it does, the TCP may optionally keep the packet queued for later processing. If not, the TCP can discard the packet. In either case the TCP can optionally acknowledge with the current left window edge.

It may happen that the process receive buffer is not present in the active memory of the HOST, but is stored on secondary storage. If this is the case, the TCP can prompt the scheduler to bring in the appropriate buffer and the packet can be queued for later processing.

If there are no more input buffers available to the TCP for temporary queueing of incoming packets, and if the TCP cannot quickly use the arriving data (e.g., a TCP to TCP message), then the packet is discarded. Assuming a sensibly functioning system, no other processes than the one for which the packet was intended should be affected by this discarding. If the delayed processing queue grows

[4] This component can serve to handle other protocols whose associated control programs are designated by internetwork destination address.

excessively long, any packets in it can be safely discarded since none of them have yet been acknowledged. Congestion at the TCP level is flexibly handled owing to the robust retransmission and duplicate detection strategy.

TCP/PROCESS COMMUNICATION

In order to send a message, a process sets up its text in a buffer region in its own address space, inserts the requisite control information (described in the following list) in a transmit control block (TCB) and passes control to the TCP. The exact form of a TCB is not specified here, but it might take the form of a passed pointer, a pseudointerrupt, or various other forms. To receive a message in its address space, a process sets up a receive buffer, inserts the requisite control information in a receive control block (RCB) and again passes control to the TCP.

In some simple systems, the buffer space may in fact be provided by the TCP. For simplicity we assume that a ring buffer is used by each process, but other structures (e.g., buffer chaining) are not ruled out.

A possible format for the TCB is shown in Fig. 11. The TCB contains information necessary to allow the TCP to extract and send the process data. Some of the information might be implicitly known, but we are not concerned with that level of detail. The various fields in the TCB are described as follows.

1) *Source Address*: This is the full net/HOST/TCP/port address of the transmitter.

2) *Destination Address*: This is the full net/HOST/TCP/port of the receiver.

3) *Next Packet Sequence Number*: This is the sequence number to be used for the next packet the TCP will transmit from this port.

4) *Current Buffer Size*: This is the present size of the process transmit buffer.

5) *Next Write Position*: This is the address of the next position in the buffer at which the process can place new data for transmission.

6) *Next Read Position*: This is the address at which the TCP should begin reading to build the next segment for output.

7) *End Read Position*: This is the address at which the TCP should halt transmission. Initially 6) and 7) bound the message which the process wishes to transmit.

8) *Number of Retransmissions/Maximum Retransmissions*: These fields enable the TCP to keep track of the number of times it has retransmitted the data and could be omitted if the TCP is not to give up.

9) *Timeout/Flags*: The timeout field specifies the delay after which unacknowledged data should be retransmitted. The flag field is used for semaphores and other TCP/process synchronization, status reporting, etc.

10) *Current Acknowledgment/Window*: The current acknowledgment field identifies the first byte of data still unacknowledged by the destination TCP.

The read and write positions move circularly around the transmit buffer, with the write position always to the left (module the buffer size) of the read position.

The next packet sequence number should be constrained to be less than or equal to the sum of the current acknowledgment and the window fields. In any event, the next sequence number should not exceed the sum of the current acknowledgment and half of the maximum possible sequence number (to avoid confusing the receiver's duplicate detection algorithm). A possible buffer layout is shown in Fig. 12.

The RCB is substantially the same, except that the end read field is replaced by a partial segment check-sum register which permits the receiving TCP to compute and remember partial check sums in the event that a segment arrives in several packets. When the final packet of the segment arrives, the TCP can verify the check sum and if successful, acknowledge the segment.

CONNECTIONS AND ASSOCIATIONS

Much of the thinking about process-to-process communication in packet switched networks has been influenced by the ubiquitous telephone system. The HOST–HOST protocol for the ARPANET deals explicitly with the opening and closing of simplex connections between processes [9],[10]. Evidence has been presented that message-based "connection-free" protocols can be constructed [12], and this leads us to carefully examine the notion of a connection.

The term *connection* has a wide variety of meanings. It can refer to a physical or logical path between two entities, it can refer to the flow over the path, it can inferentially refer to an action associated with the setting up of a path, or it can refer to an association between two or more entities, with or without regard to any path between them. In this paper, we do not explicitly reject the term connection, since it is in such widespread use, and does connote a meaningful relation, but consider it exclusively in the sense of an association between two or more entities without regard to a path. To be more precise about our intent, we shall define the relationship between two or more ports that are in communication, or are prepared to communicate to be an *association*. Ports that are associated with each other are called *associates*.

It is clear that for any communication to take place between two processes, one must be able to address the other. The two important cases here are that the destination port may have a global and unchanging address or that it may be globally unique but dynamically reassigned. While in either case the sender may have to learn the destination address, given the destination name, only in the second instance is there a requirement for learning the address from the destination (or its representative) each time an association is desired. Only after the source has learned how to address the destination can an association be said to have occurred. But this is not yet sufficient. If

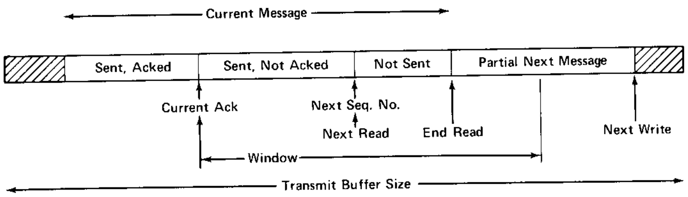

Fig. 12. Transmit buffer layout.

ordering of delivered messages is also desired, both TCP's must maintain sufficient information to allow proper sequencing. When this information is also present at both ends, then an association is said to have occurred.

Note that we have not said anything about a path, nor anything which implies that either end be aware of the condition of the other. Only when both partners are prepared to communicate with each other has an association occurred, and it is possible that neither partner may be able to verify that an association exists until some data flows between them.

CONNECTION-FREE PROTOCOLS WITH ASSOCIATIONS

In the ARPANET, the interface message processors (IMP's) do not have to open and close connections from source to destination. The reason for this is that connections are, in effect, always open, since the address of every source and destination is never[5] reassigned. When the name and the place are static and unchanging, it is only necessary to label a packet with source and destination to transmit it through the network. In our parlance, every source and destination forms an association.

In the case of processes, however, we find that port addresses are continually being used and reused. Some ever-present processes could be assigned fixed addresses which do not change (e.g., the logger process). If we supposed, however, that every TCP had an infinite supply of port addresses so that no old address would ever be reused, then any dynamically created port would be assigned the next unused address. In such an environment, there could never be any confusion by source and destination TCP as to the intended recipient or implied source of each message, and all ports would be associates.

Unfortunately, TCP's (or more properly, operating systems) tend not to have an infinite supply of internal port addresses. These internal addresses are reassigned after the demise of each port. Walden [12] suggests that a set of unique uniform external port addresses could be supplied by a central registry. A newly created port could apply to the central registry for an address which the central registry would guarantee to be unused by any HOST system in the network. Each TCP could maintain tables matching external names with internal ones, and use the external ones for communication with other processes. This idea violates the premise that interprocess communication should not require centralized control. One would have to extend the central registry service to include all HOST's in all the interconnected networks to apply this idea to our situation, and we therefore do not attempt to adopt it.

Let us consider the situation from the standpoint of the TCP. In order to send or receive data for a given port, the TCP needs to set up a TCB and RCB and initialize the window size and left window edge for both. On the receive side, this task might even be delayed until the first packet destined for a given port arrives. By convention, the first packet should be marked so that the receiver will synchronize to the received sequence number.

On the send side, the first request to transmit could cause a TCB to be set up with some initial sequence number (say, zero) and an assumed window size. The receiving TCP can reject the packet if it wishes and notify the sending TCP of the correct window size via the acknowledgment mechanism, but only if either

1) we insist that the first packet be a complete segment;
2) an acknowledgment can be sent for the first packet (even if not a segment, as long as the acknowledgment specifies the next sequence number such that the source also understands that no bytes have been accepted).

It is apparent, therefore, that the synchronizing of window size and left window edge can be accomplished without what would ordinarily be called a connection setup.

The first packet referencing a newly created RCB sent from one associate to another can be marked with a bit which requests that the receiver synchronize his left window edge with the sequence number of the arriving packet (see SYN bit in Fig. 8). The TCP can examine the source and destination port addresses in the packet and in the RCB to decide whether to accept or ignore the request.

Provision should be made for a destination process to specify that it is willing to LISTEN to a specific port or "any" port. This last idea permits processes such as the logger process to accept data arriving from unspecified sources. This is purely a HOST matter, however.

The initial packet may contain data which can be stored or discarded by the destination, depending on the availability of destination buffer space at the time. In the other direction, acknowledgment is returned for receipt of data which also specifies the receiver's window size.

If the receiving TCP should want to reject the synchronization request, it merely transmits an acknowledgment carrying a release (REL) bit (see Fig. 8) indicating that the destination port address is unknown or inaccessible. The sending HOST waits for the acknowledgment (after accepting or rejecting the synchronization request) before sending the next message or segment. This rejection is quite different from a negative data acknowledgment. We do not have explicit negative acknowledgments. If no acknowledgment is returned, the sending HOST may

[5] Unless the IMP is physically moved to another site, or the HOST is connected to a different IMP.

retransmit without introducing confusion if, for example, the left window edge is not changed on the retransmission.

Because messages may be broken up into many packets for transmission or during transmission, it will be necessary to ignore the REL flag except in the case that the EM flag is also set. This could be accomplished either by the TCP or by the GATEWAY which could reset the flag on all but the packet containing the set EM flag (see Fig. 9).

At the end of an association, the TCP sends a packet with ES, EM, and REL flags set. The packet sequence number scheme will alert the receiving TCP if there are still outstanding packets in transit which have not yet arrived, so a premature dissociation cannot occur.

To assure that both TCP's are aware that the association has ended, we insist that the receiving TCP respond to the REL by sending a REL acknowledgment of its own.

Suppose now that a process sends a single message to an associate including an REL along with the data. Assuming an RCB has been prepared for the receiving TCP to accept the data, the TCP will accumulate the incoming packets until the one marked ES, EM, REL arrives, at which point a REL is returned to the sender. The association is thereby terminated and the appropriate TCB and RCB are destroyed. If the first packet of a message contains a SYN request bit and the last packet contains ES, EM, and REL bits, then data will flow "one message at a time." This mode is very similar to the scheme described by Walden [12], since each succeeding message can only be accepted at the receiver after a new LISTEN (like Walden's RECEIVE) command is issued by the receiving process to its serving TCP. Note that only if the acknowledgment is received by the sender can the association be terminated properly. It has been pointed out[6] that the receiver may erroneously accept duplicate transmissions if the sender does not receive the acknowledgment. This may happen if the sender transmits a duplicate message with the SYN and REL bits set and the destination has already destroyed any record of the previous transmission. One way of preventing this problem is to destroy the record of the association at the destination only after some known and suitably chosen timeout. However, this implies that a new association with the same source and destination port identifiers could not be established until this timeout had expired. This problem can occur even with sequences of messages whose SYN and REL bits are separated into different internetwork packets. We recognize that this problem must be solved, but do not go into further detail here.

Alternatively, both processes can send one message, causing the respective TCP's to allocate RCB/TCB pairs at both ends which rendezvous with the exchanged data and then disappear. If the overhead of creating and destroying RCB's and TCB's is small, such a protocol might be adequate for most low-bandwidth uses. This idea might also form the basis for a relatively secure transmission system. If the communicating processes agree to change their external port addresses in some way known only to each other (i.e., pseudorandom), then each message will appear to the outside world as if it is part of a different association message stream. Even if the data is intercepted by a third party, he will have no way of knowing that the data should in fact be considered part of a sequence of messages.

We have described the way in which processes develop associations with each other, thereby becoming associates for possible exchange of data. These associations need not involve the transmission of data prior to their formation and indeed two associates need not be able to determine that they are associates until they attempt to communicate.

[6] S. Crocker of ARPA/IPT.

CONCLUSIONS

We have discussed some fundamental issues related to the interconnection of packet switching networks. In particular, we have described a simple but very powerful and flexible protocol which provides for variation in individual network packet sizes, transmission failures, sequencing, flow control, and the creation and destruction of process-to-process associations. We have considered some of the implementation issues that arise and found that the proposed protocol is implementable by HOST's of widely varying capacity.

The next important step is to produce a detailed specification of the protocol so that some initial experiments with it can be performed. These experiments are needed to determine some of the operational parameters (e.g., how often and how far out of order do packets actually arrive; what sort of delay is there between segment acknowledgments; what should be retransmission timeouts be?) of the proposed protocol.

ACKNOWLEDGMENT

The authors wish to thank a number of colleagues for helpful comments during early discussions of international network protocols, especially R. Metcalfe, R. Scantlebury, D. Walden, and H. Zimmerman; D. Davies and L. Pouzin who constructively commented on the fragmentation and accounting issues; and S. Crocker who commented on the creation and destruction of associations.

REFERENCES

[1] L. Roberts and B. Wessler, "Computer network development to achieve resource sharing," in *1970 Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 543–549.

[2] L. Pouzin, "Presentation and major design aspects of the CYCLADES computer network," in *Proc. 3rd Data Communications Symp.*, 1973.

[3] F. R. E. Dell, "Features of a proposed synchronous data network," in *Proc. 2nd Symp. Problems in the Optimization of Data Communications Systems*, 1971, pp. 50–57.

[4] R. A. Scantlebury and P. T. Wilkinson, "The design of a switching system to allow remote access to computer services by other computers and terminal devices," in *Proc. 2nd Symp. Problems in the Optimization of Data Communications Systems*, 1971, pp. 160–167.

[5] D. L. A. Barber, "The European computer network project," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 192–200.

[6] R. Despres, "A packet switching network with graceful saturated operation," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 345–351.

[7] R. E. Kahn and W. R. Crowther, "Flow control in a resource-sharing computer network," *IEEE Trans. Commun.*, vol. COM-20, pp. 539–546, June 1972.

[8] J. F. Chambon, M. Elie, J. Le Bihan, G. LeLann, and H. Zimmerman, "Functional specification of transmission station in the CYCLADES network. ST-ST protocol" (in French), I.R.I.A. Tech. Rep. SCH502.3, May 1973.

[9] S. Carr, S. Crocker, and V. Cerf, "HOST-HOST Communication Protocol In the ARPA Network," in *Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 589–597.

[10] A. McKenzie, "HOST/HOST protocol for the ARPA network," in *Current Network Protocols*, Network Information Cen., Menlo Park, Calif., NIC 8246, Jan. 1972.

[11] L. Pouzin, "Address format in Mitranet," NIC 14497, INWG 20, Jan. 1973.

[12] D. Walden, "A system for interprocess communication in a resource sharing computer network," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 221–230, Apr. 1972.

[13] B. Lampson, "A scheduling philosophy for multiprocessing systems," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 347–360, May 1968.

[14] F. E. Heart, R. E. Kahn, S. Ornstein, W. Crowther, and D. Walden, "The interface message processor for the ARPA computer network," in *Proc. Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 551–567.

[15] N. G. Anslow and J. Hanscoff, "Implementation of international data exchange networks," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 181–184

[16] A. McKenzie, "HOST/HOST protocol design considerations," INWG Note 16, NIC 13879, Jan. 1973.

[17] R. E. Kahn, "Resource-sharing computer communication networks," *Proc. IEEE*, vol. 60, pp. 1397–1407, Nov. 1972.

[18] Bolt, Beranek, and Newman, "Specification for the interconnection of a host and an IMP," Bolt Beranek and Newman, Inc., Cambridge, Mass., BBN Rep. 1822 (revised), Apr. 1973.

Vinton G. Cerf was born in New Haven, Conn., in 1943. He did undergraduate work in mathematics at Stanford University, Stanford, Calif., and received the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, Calif., in 1972.

He was with IBM in Los Angeles from 1965 through 1967 and consulted and/or worked part time at UCLA from 1967 through 1972. Currently he is Assistant Professor of Computer Science and Electrical Engineering at Stanford University, and consultant to Cabledata Associates. Most of his current research is supported by the Defense Advanced Research Projects Agency and by the National Science Foundation on the technology and economics of computer networking. He is Chairman of IFIP TC6.1, an international network working group which is studying the problem of packet network interconnection.

★

Robert E. Kahn (M'65) was born in Brooklyn, N. Y., on December 23, 1938. He received the B.E.E. degree from the City College of New York, New York, in 1960, and the M.A. and Ph.D. degrees from Princeton University, Princeton, N. J., in 1962 and 1964, respectively.

From 1960 to 1962 he was a Member of the Technical Staff of Bell Telephone Laboratories, Murray Hill, N. J., engaged in traffic and communication studies. From 1964 to 1966 he was a Ford Postdoctoral Fellow and an Assistant Professor of Electrical Engineering at the Massachusetts Institute of Technology, Cambridge, where he worked on communications and information theory. From 1966 to 1972 he was a Senior Scientist at Bolt Beranek and Newman, Inc., Cambridge, Mass., where he worked on computer communications network design and techniques for distributed computation. Since 1972 he has been with the Advanced Research Projects Agency, Department of Defense, Arlington, Va.

Dr. Kahn is a member of Tau Beta Pi, Sigma Xi, Eta Kappa Nu, the Institute of Mathematical Statistics, and the Mathematical Association of America. He was selected to serve as a National Lecturer for the Association for Computing Machinery in 1972.