#201

The Influence Of Control Procedures On The Performance of Packet-Switched Networks

HOLGER OPDERBECK and LEONARD KLEINROCK
University of California at Los Angeles, California 90024

Abstract

The ARPANET experience has produced occasional network deadlocks and other (less catastrophic) forms of network degradation. These problems have been traced to logical faults in the network flow control procedure. In this paper we discuss the general aims and problems of flow control, describe aspects of the ARPANET approach, and then outline some of the deadlocks and degradations which have been discovered. No satisfactory solution to the general problem of discovering hidden deadlocks is available yet.

I. Introduction

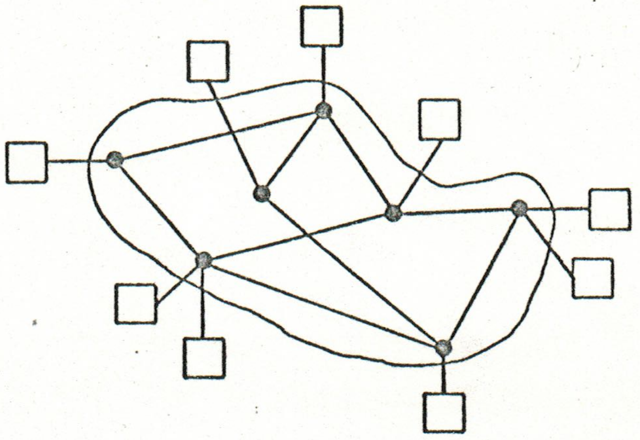
Whenever two information processing systems exchange data, carefully designed control procedures are necessary to insure safe and correct transfer of the information. The main purpose of these control procedures is:

1. prevention of loss of data (for example, the receiving system must signal its readiness to accept the data)
2. duplicate detection (each transferred information item should be read only once by the receiving system)
3. error detection and correction
4. efficient use of physical resources
5. deadlock prevention

There are other issues involved in the data transfer such as recovery from failures, security, and accounting that are not dealt with in this paper.

If the two systems are in close physical proximity a carefully designed handshake procedure usually provides most of the control that is necessary for the correct transfer of data. If two (or more) systems are connected over telephone lines, special control messages are usually exchanged between the systems to control the flow of data. The most complex case arises when a communications network is used to inter-connect a variety of different information processing systems such as computer terminals, remote job entry stations, sensors, speech processors, etc. (see Figure 1). In this case, there are several levels of data transfer which need to be controlled. In particular, the flow of data has to be controlled between:

a) any pair of communications processors
b) a communications processor and the attached information processing systems
c) any pair of information processing systems
d) any pair of communicating processes if the information processing systems are multiprogramming or time-sharing computer systems



● COMMUNICATIONS PROCESSOR
□ INFORMATION PROCESSING SYSTEM

Figure 1. Computer—Communication Network Structure.

In this paper we will mostly deal with a) and b), that is, the control procedures within and at the boundary of the communications network. We will focus our attention on packet-switched networks and discuss the impact of flow control procedures on the performance of these networks. To demonstrate the difficulty of designing "correct" control procedures we will report on the experience with the operation of the ARPANET [1,2,3,4]. Several design oversights in the ARPANET are discussed which caused or could have caused serious performance degradations or even a complete lockup. These conditions have since been identified and corrected.

II. Performance Criteria

Before we go into a discussion of various control procedures, we will first describe the performance criteria that are used to evaluate packet-switched computer communication networks. In this context, we are interested in performance measures which describe the ability of the communications network to accommodate different types of traffic. (This excludes performance measures like reliability, availability, flexibility, etc.). If the communications network were to carry only one type of traffic, one performance measure would probably be enough to describe the performance of the system. For example, if the main purpose of the communications network were to provide high throughput paths, the maximum throughput between any pair of nodes under various conditions would be a proper measure of its performance. However, most communication networks are designed to accommodate several kinds of traffic. Therefore several performance measures are essential to understand the performance characteristics of such a network.

We will distinguish the following three types of traffic:

1. High Throughput Traffic (HT-traffic)
2. Low Delay Traffic (LD-traffic)
3. Real Time Traffic (RT-traffic)

HT-traffic is required for the transmission of large files of data. In this case the total time between the initiation and the completion of a file transfer needs to be minimized. The delay of individual packets between source and destination node, however, is less important. Therefore HT-packets should not necessarily be sent over the path of minimum expected delay but over the path of maximum excess capacity. Extensive buffering is usually used to increase the line utilization and therefore the throughput. Remote job entry represents a typical application which requires HT-traffic.

LD-traffic is typically encountered during the interactive use of computers. Here the delay for individual messages between the source and destination node needs to be minimized. LD-traffic can be characterized by a large peak to average line utilization ratio ("bursty" traffic). In contrast to HT-traffic, queueing delays due to buffering of messages are undesirable. The remote use of time-sharing systems represents a typical application which requires LD-traffic.

RT-traffic has characteristics of both, HT-traffic and LD-traffic. The transmission of digitized speech represents the best example for a data transmission which requires RT-traffic. In this case it is important that the delay for a large percentage of messages be less than some tolerable threshold value. Also, the network should be able to maintain a constant level of throughput. A characteristic property of RT-traffic is the redundant encoding of the information it carries. Therefore the control procedures need not be as much concerned about the prevention or loss of data as is the case with HT-traffic or LD-traffic.

If the purpose of the network is to carry only one type of traffic, it is relatively easy to design efficient control procedures. It is much more difficult to design control procedures which satisfy the requirements of two types of traffic simultaneously. If all three types of traffic need to be accommodated in the network, one usually has to compromise. The kind of compromise can be described graphically by the performance triangle of Figure 2 which was first

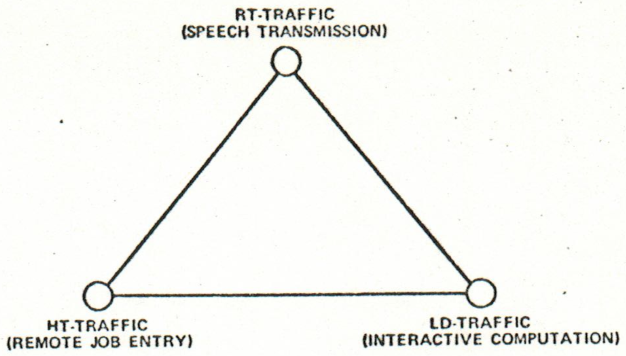


Figure 2. Performance Triangle.

suggested to the authors by Danny Cohen of USC-ISI. Each point inside the triangle represents a possible compromise. A design which optimizes the network for one type of traffic will often be suboptimal for the other two types of traffic.

If the purpose of the communications network is to carry LD-traffic as well as HT-traffic, a well-known technique to accommodate both types of traffic is to use different priority levels. The LD-packets are given a higher priority such that HT-packets are only served when the LD-packet queue is empty. If RT-packets are also to be transmitted we have a more difficult situation. Because of the nature of the traffic, it is not clear whether RT-packets should have a higher or a lower priority than LD-packets. In this case, priorities which increase as a function of the incurred delay appear to provide a more satisfactory solution. For example, the RT-packets could start out with a lower priority than the LD-packets. However, the priority of those RT-packets which incur extremely large delays could be increased in such a way that these packets still reach their destination within the specified time interval [5].

All these considerations apply to the management of output queues, i.e., only packets which have already been accepted by a node are involved in the decision. However, there is an equally important decision to be made concerning the admission of packets to a node. Networks cannot afford to accept all the traffic that is offered without some control. This would inevitably lead to heavy congestion and possibly lockup of the entire network. There must be rules which govern the acceptance of traffic from the outside. These rules are commonly known as flow control procedures. The next section will introduce different kinds of flow control procedures and discuss their influence on the different types of traffic.

III. Flow Control Procedures

There are two types of flow control: local flow control and global flow control. Local flow control is an important characteristic of any packet-switched network. It is a direct consequence of the limited buffer space in each node. Whenever this buffer space is used up, a node has to stop further input from outside the net and from its neighbors. To avoid congestion, the input is usually stopped even before all the buffers are occupied. For example, there is only a limited number of packets allowed on each output queue. Packets which would have to join a full output queue are rejected. There may also be a limit on the total number of "store-and-forward" buffers such that only packets which leave the network at the node in question ("reassembly packets" 0 are accepted if all store-and-forward buffers are occupied [2,6].

The existence of local flow control procedures implies that packets may experience an admission delay which will be non-zero in case of (local) congestion. This admission delay is experienced not only by packets which enter the net from the outside but also by packets which are transmitted from a neighboring node. It is important to realize that this admission delay contributes to the total delay in the same way as the queueing delay in the output queues. Therefore the decision as to what packets to accept or reject should be based on a similar priority structure as that described for the management of output queues. For example, HT-packets could already be rejected while LD-packets and RT-packets are still being accepted. Also, for packets with the same priority, preference should be given to packets which are transmitted from a neighboring node over those which try to enter the net from the outside. This strategy helps effectively to prevent further congestion [7].

Local flow control alone is not sufficient to avoid congestion in a packet-switched network. There also needs to be some limitation on the total number of packets which can be handled by the network simultaneously. Procedures which achieve this limitation are called global flow control procedures. If the global flow control works properly, further input to the communications network is stopped well before all the buffer space in the net is occupied. There are two methods of global flow control which have been investigated: end-to-end flow control (ARPA-like[6]) and isarithmic flow control (NPL-like[8]).

In the current ARPANET there is a maximum number of four messages which can be outstanding between any pair of source-destination nodes (or IMPs). This scheme has two major advantages:
1. It is easy to implement (Whenever four end-to-end acknowledgment packets, called RFNMs, are outstanding, further input from any of the attached computers, called HOSTs, is stopped).
2. The input to the net is stopped rapidly when a destination IMP or HOST goes down.

A disadvantage of the current scheme is that the flows between distinct pairs of HOSTs which are connected to the same source and destination IMPs suffer from interference. Therefore a change of the global flow control from an IMP-to-IMP to a HOST-to-HOST basis is currently being considered. There are other areas of possible improvement of the end-to-end global flow control procedure which are worth investigating. For example, the limit of four outstanding messages appears to be rather arbitrary. Ideally, this number should change dynamically as a function of the total network load. It should also be considered to limit the number of outstanding packets, not messages since most of the network resources are allocated to packets, and not messages. This number of outstanding packets may then depend on the minimum number of hops between source and destination IMP. This is in contrast to the current scheme which is distance-independent.

Isarithmic flow control is based on the idea of a fixed number of "containers". A packet must be placed in an empty container before being transmitted. When the packet arrives at its destination node, it is removed from the container which then becomes available for the transmission of another packet. The empty container may be stored at the node where it became empty or be sent to a neighboring node. In the isarithmic flow control scheme, the admission delay is the time a packet waits for an empty container to become available.

It is important to realize that with this flow control method the management of the empty containers has a direct influence on the performance of the network. For example, if the network is to be optimized for LD-traffic, it is desirable to restrict the movement of the empty containers in such a way that there are always a few empty containers waiting at each node. This guarantees a zero admission delay. If the network is, on the other hand, to be optimized for HT-traffic, a node should have the ability to acquire empty containers from its neighbors. However, if a large percentage of the containers is used by a few nodes to achieve high throughput, other nodes will be out of empty containers. This will introduce considerable admission delays at these nodes. This example shows clearly that HT-traffic and LD-traffic are conflicting goals in this environment.

A possibility to reconcile the conflicting interests is to provide for different types of containers. Some containers may, for instance, only be allowed to transport LD-packets. These are then managed separately to suboptimize the LD-traffic. Another type of container may be allowed to transport HT-packets and LD-packets. These containers can be used to achieve a high throughput. Similar observations can be made if HT-traffic, LD-traffic, and RT-traffic must be accommodated in the same network.

IV. Lockups and Throughput Degradation in the ARPANET

4.1 Previous Lockup Problems

Lockup or deadlock conditions are one of the most serious system malfunctions that can occur in a computer system or network. Communication protocols have to be designed very carefully to avoid the occurrence of these lockups. Their common characteristic is that they occur only under unusual circumstances which were not foreseen or deemed too unlikely to occur by the protocol designers. (However, these designers often are not the ones in a position to evaluate such likelihoods quantitatively.)

In the ARPANET, HOSTs communicate with each other via a sequence of messages. An IMP takes in a message from its HOST, forms it into packets, and ships the packets separately into the network. A message consists of up to eight packets whose maximum size is approximately 1000 bits. The destination IMP reassembles the packets and delivers them in sequence to the receiving HOST. This reassembly of packets in conjunction with the early form of flow control caused the best known lockup that has occurred in the ARPANET. The repeated observation of this "reassembly lockup" led to a major redesign of the ARPANET flow control mechanism.

Reassembly lockup could occur in the subnet when reassembly space was unavailable to store incoming multipacket messages. Let us assume that all the reassembly buffers at some IMP A are either occupied or reserved for awaited packets of partially reassembled messages. Reassembly lockup occurred when all the neighbors of IMP A were filled with packets also headed to A which IMP A could not accept, thereby preventing packets at other IMPs from reaching the destination A and completing the partially reassembled messages [9].

Direct store-and-forward lockup is another example of a lockup which can occur in a packet-switched network if no proper precautions are taken [9]. Let us assume that all store-and-forward buffers in some IMP A are filled with packets headed to the neighboring IMP B and that all store-and-forward buffers in IMP B are filled with packets headed to IMP A. Since there is no store-and-forward buffer space available in either IMP, no packet can be successfully transmitted between these two IMPs and a deadlock situation results. There is, of course, an easy way to remedy this situation. One has to make sure that not all of the store-and-forward buffers can reside on a single output queue. In the ARPANET, only 8 of the 20 store-and-forward buffers can be placed on a single output queue.

Indirect store-and-forward lockup can occur when all the store-and-forward buffers in a loop of IMPs become filled with packets which all travel in the same direction (clockwise or counter-clockwise) [9]. Although such a highly structured traffic pattern is very unlikely to occur it can be shown that, for the lockup to establish itself, this undesirable packet flow need only persist for about 1 sec. It appears to be difficult to find an efficient procedure which prevents indirect store-and-forward lockup from occurring. In this case it may be more efficient to provide for the recovery from the lockup than for its prevention. For this purpose a priority structure on the admission of packets as described in Section 3 is very helpful since, for example, it may allow the exchange of control and LD-packets while HT-packets are already locked up.

The last lockup in the ARPANET which caused a revision of the flow control procedure happened at the end of 1973 ("Christmas lockup"). This dormant lockup condition was brought to light by collecting snapshot measurement messages at UCLA from all sites simultaneously. The Christmas lockup happened when snapshot messages arrived at the UCLA IMP which had allocated reassembly storage for them and no reassembly blocks were free. (A reassembly block is a piece of storage used in the actual process of reassembling packets back into messages.) To avoid this kind of lockup reassembly blocks are now allocated along with the reassembly buffers for each multiple-packet message [10].

Aside from these flow control lockups, a number of hardware failures have produced severe lockup conditions. This has led to the extensive use of software checksums to protect data packets and sensitive pieces of code [11].

4.2 Piggyback Lockup

As long as it is not possible to design practical communication protocols which guarantee deadlock-free operation it is vital to continually check those protocols that are currently in use for any such failures - even if they appear "very unlikely" to occur. In this section we comment on a possible deadlock condition in the IMP subnet which, to our knowledge, has not yet occurred, neither had it previously been identified. Though we have never seen this problem actually happen it may occur in the future if no precautions are taken. This possible lockup condition is due to the sequencing of messages in the subnet.

As mentioned before, the flow control mechanism in the ARPANET was modified in some significant ways to avoid the occurrence of reassembly lockup [6]. Specifically, no multi-packet message is allowed to enter the network until storage for the message has been allocated at the destination IMP. As soon as the source IMP takes in the first packet of a multi-packet message, it sends a small control message to the destination IMP requesting that reassembly storage be reserved. It does not take in further packets from the HOST until it receives an allocation message in reply.

To maximize the effective bandwidth for sequences of long messages, the end-to-end acknowledgment message ("RFNM" for request-for-next-message) may carry a storage allocation (the piggybacked ALLOCATE). If the source HOST delays too long, or if the data transfer is complete, the source IMP returns the unused allocation to the destination IMP.

To guarantee that messages leave the destination IMP in the same order as they entered the source IMP, each message carries a sequence number. This sequencing of messages has the potential of introducing deadlock conditions. The reason for this is that any message, say MSG(n+1), which is out of order (and therefore cannot be delivered to its destination HOST) may use up resources that are required by MSG(n) which must be delivered next. Therefore, MSG(n) may not be able to reach its destination IMP which, in turn, prevents the other messages (n+1, etc.) that are out of order from being delivered to their destination HOST(s). For this reason one has to be very careful not to allocate too many resources (e.g. buffers) to messages that are out of order.

To avoid lockup conditions the current flow control procedure in the ARPANET takes the two following precautions:

1. Requests for buffer allocation are always serviced in order of message number; i.e. no 'ALLOCATE' is returned for MSG(n+1) if MSG(n) (or a request for buffer allocation for MSG(n)) has not yet been

received and serviced.

2. Single packet messages (regular and priority) that arrive at the destination IMP out of order are not accepted unless they were retransmitted in response to a previous buffer allocation. These messages are treated rather as a request for the allocation of one buffer (according to 1 above) and the message text is discarded.

With these two precautions the occurrence of deadlock conditions appears to be impossible. However, there is a second buffer allocation mechanism that is not tied to the message sequencing, namely, the ALLOCATE that is piggybacked on the RFNM for a multiple-packet message. The piggybacked ALLOCATE represents a buffer allocation for the next multiple-packet message, and not for the next message in sequence. Thus, if the next message in sequence is a single-packet message, the piggybacked ALLOCATE in effect allocates buffers to a message that is out of order.

Let us see how this situation can lead to a deadlock condition. Assume there is a maximum number of 24 reassembly buffers in each IMP. Let IMPs A, B, and C continually transmit 8-packet messages to the same destination IMP D such that all 24 reassembly buffers in IMP D are used up by this transmission of multiple-packet messages. If now, in the stream of 8-packet messages, IMP A sends a single-packet message it will generally not be accepted by destination IMP D since there is no reassembly buffer space available. (There may be a free reassembly buffer if the single-packet message just happens to arrive during the time one of the three 8-packet messages is being transmitted to its HOST). The single-packet message will therefore be treated as a request for buffer allocation. This request will not be serviced before the RFNM of the previous multiple-packet message has been sent. At this time, however, all the free reassembly buffers have already been allocated to the next multiple-packet message via the piggybacked ALLOCATE mechanism. The only chance for the single-packet message to get its allocation request satisfied is to grab a reassembly buffer from one of the other two 8-packet messages. This attempt may be unsuccessful because it depends on the timing of events in the IMP. A deadlock condition can occur if IMP B and IMP C also send a single-packet message in their stream of 8-packet messages which cannot be serviced for the same reason. In this case, the three 8-packet messages that will arrive later at IMP D cannot be delivered to their destination HOST(s) because they are out of order. The three single-packet messages that should be delivered next, however, will never reach the destination IMP since there is no reassembly space available. Table 1 shows a possible sequence of events that leads to this deadlock condition. Note that an ALLOCATE for one of the single-packet messages A1, B1, and C1 can only be returned to source IMP A, B, and C, respectively, after the RFNM (with its piggybacked ALLOCATE) for the previous 8-packet message has been sent. If these RFNMs are sent in sequence, i.e. without an ALLOCATE for one of the single-packet messages in between, the temporarily freed reassembly storage (events (7) through (9)) is implicitly allocated to the next multiple-packet messages (events (10) through (12)) and not to any of the single-packet messages. The next 8-packet messages are, however, out of order and cannot be delivered to their destination HOST(s).

It appears as though such a lockup can only occur if the number of reassembly buffers is a

Table 1 Example for the Piggyback Lockup

		# of allocated reassembly buffers	# of reassembly buffers in use	# of free reassembly buffers
	Initially	24	0	0
1.	A8 arrives	16	8	0
2.	B8 arrives	8	16	0
3.	C8 arrives	0	24	0
4.	A1 arrives	0	24	0
5.	B1 arrives	0	24	0
6.	C1 arrives	0	24	0
7.	A8 complete	0	16	8
8.	B8 complete	0	8	16
9.	C8 complete	0	0	24
10.	A8 RFNM/ALL	8	0	16
11.	B8 RFNM/ALL	16	0	8
12.	C8 RFNM/ALL	24	0	0
13.	A8 arrives	16	8	0
14.	B8 arrives	8	16	0
15.	C8 arrives	0	24	0
16.	- deadlock -			

Explanation of notation:

event: A8 arrives	all 8 packets of the 8-packet message from IMP A have arrived at IMP D
event: C1 arrives	a single packet message from IMP C has arrived at IMP D (and is treated as a request for buffer allocation)
event: B8 complete	the last packet of the 8-packet message from IMP B has been received by its destination HOST
event: A8 RFNM/ALL	a RFNM with the piggybacked ALLOCATE is sent to IMP A

multiple of eight. Indeed, the probability of a lockup in this latter case is much higher. However, deadlocks can also occur if the number of reassembly buffers is not a multiple of eight. Let us assume there are 26 instead of 24 reassembly buffers. Assume also that, due to alternate paths or line failure, the second packet of a 2-packet message arrives at IMP D before a single-packet message from the same source IMP A. The single-packet message has a smaller sequence number and must therefore be delivered to its destination HOST before the 2-packet message. When the second packet of the 2-packet message arrives at IMP D the IMP realizes that only 2 of the allocated 8 buffers will be needed. Therefore 6 buffers are returned to the pool of free reassembly buffers. If there were 26-3x8=2 buffers in the pool before, the pool size is increased by 6 to 8 buffers. These 8 buffers, however, are just enough to send out another piggybacked ALLOCATE. The single-packet message may therefore find the pool of free reassembly buffers empty although the total number of reassembly buffers is not a multiple of eight. The 2-packet message cannot be delivered to its destination HOST because it is out of order. Therefore the deadlock condition we described before may occur again.

We agree that the above mentioned sequence of events is unlikely to occur (otherwise one would have observed it already). This is particularly true since the current maximum number of reassembly buffers (42) is much larger than 24. Judging from past experience with computer systems and networks, however, we know that even very unlikely events have a tendency to occur in the long run. Also, the probability of this deadlock condition increases with increasing traffic in the net. Therefore, it is certainly worthwhile to modify the flow control in such a way that this deadlock cannot occur. It turns out that a minor modification already achieves the desired effect. Recall that the described deadlock can only occur because single- and multiple-packet messages use the same pool of reassembly buffers. If we set aside a single reassembly buffer (or one for each destination HOST) that can be used only by single-packet messages this lockup condition due to message sequencing cannot occur.

4.3 Throughput Degradation

As pointed out before, the throughput and delay requirements for RT-traffic are quite different from the throughput and delay requirements for interactive use or file transfers. For the transmission of digitized speech, for instance, it is necessary to achieve a relatively high throughput for small messages since long messages result in long source delays to fill the large buffers. We realize that up to now little attempt was made to optimize the transmission of RT-traffic in the ARPANET. It was nevertheless surprising for us to find out that the observed throughput for single-packet messages is in many cases only about one fourth of what one would expect. In what follows we are going to explain why this happens and what could be done to correct this situation.

As mentioned before, single-packet messages are not accepted by the destination IMP if they arrive out of order. They are rather treated as a request for the allocation of one reassembly buffer. The corresponding ALLOCATE is then sent back to the source IMP only after the RFNM for the previous message has been processed. We therefore may have the following sequence of events:

1. MSG(i) sent from SOURCE-IMP (message i is sent from the source IMP to the destination IMP).
2. MSG(i+1) sent from SOURCE-IMP.
3. MSG(i+1) arrives at DEST-IMP (due to an alternate path or a line error, MSG(i+1) arrives at the destination IMP out of order; it is treated as a request for one reassembly buffer allocation and then discarded).
4. MSG(i) arrives at DEST-IMP (MSG(i) arrives at the destination IMP; it is put on the proper HOST output queue).
5. RFNM(i) sent from DEST-IMP (after MSG(i) has been accepted by the destination HOST the RFNM is sent to the source IMP).
6. ALL(i+1) sent from DEST-IMP (only after the RFNM for MSG(i) has been processed can the ALLOCATE for MSG(i+1)be sent).
7. RFNM(i) arrives at SOURCE-IMP.
8. ALL(i+1) arrives at SOURCE-IMP.
9. MSG(i+1) is retransmitted from SOURCE-IMP.
10. MSG(i+1) arrives at DEST-IMP (now MSG(i+1) is put on the proper HOST output queue).
11. RFNM(i+1) sent from DEST-IMP.
12. RFNM(i+1) arrives at SOURCE-IMP.

Figure 3 describes this sequence of events graphically. Note that the round-trip time for MSG(i+1) is the time interval between event 2 and event 12. The round-trip time for MSG(i+1)is more than twice as large as it would have been if it had arrived in order, other conditions being unchanged. Therefore a line error will in many cases not only delay the message in error but also the next single-packet message if this message follows the preceding message within 125 msec, the error retransmission timeout interval. Also, a faster, alternate path to the destination IMP can actually slow down the transmission since it causes messages to arrive there out of order.

This situation becomes even worse when we consider RFNM-driven single-packet message

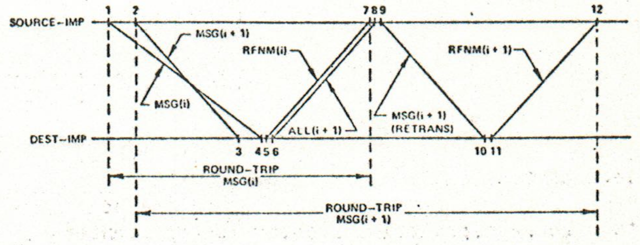


Figure 3. Retransmission Delay for MSG(i + 1).

traffic. Table 2 shows a possible sequence of events. We again assume that MSG(i+1) reaches the destination IMP before MSG(i). Since the traffic is RFNM-driven, the arrival of RFNM(i), RFNM(i+1),.... is followed by the sending of MSG(i+4),MSG(i+5),....

The most interesting fact about this sequence of events is that the arrival of MSG(i+1) before MSG(i) at the destination IMP causes not only MSG(i+1) but all future messages to be retransmitted--though we do not assume that any of the future messages arrive out of order. The table also shows that the round-trip time for MSG(i+4) and all future messages is more than four times as large as it would be without these undesirable retransmissions. It is also noteworthy that, once this retransmission pattern has established itself, there is almost no way the system can recover from this condition other than interrupting the input stream at the source IMP. A single arrival out of order of any of the later user or control messages, for instance, will not change this retransmission pattern. The normal flow of single-packet messages will reestablish itself if, for example, MSG(i+4), MSG(i+5), and MSG(i+6) are simultaneously delayed for several hundred milliseconds such that MSG(i+1), MSG(i+2), and MSG(i+3) can be retransmitted in the meantime. The probability of occurrence of such an event is, however, extremely small. Therefore one can consider the system as being trapped in this undesirable retransmission condition. The "normal" flow of messages, on the other hand, represents only the transient behavior of the system since there is always a finite probability that two messages arrive out of order due to transmission errors.

As mentioned before, the system can only recover from this throughput (and delay) degradation if the input stream of single-packet messages is interrupted. In case of speech transmission, however, this might not occur for some time. Therefore speech transmission systems would in many cases have to work with only one fourth of the expected single-packet bandwidth. Since this is clearly an unacceptable condition we now describe a method that could be used to avoid the undesirable retransmission of messages.

Recall that a single-packet message is rejected at the destination IMP and later retransmitted if the RFNM for the preceding message has not yet been sent to the source IMP. This is mainly done to prevent the occurrence of reassembly lockup conditions. Therefore the problem cannot be solved by simply accepting all single-packet messages without additional measures to prevent deadlocks. This could lead to a reassembly lockup if a large number of single-packet messages from several source IMPs arrives at their common destination IMP out of order. In this case the destination IMP might not be able to accept those messages that are in order because of the lack of reassembly buffers. As a result the system is deadlocked. Any solution of the throughput degradation problem must guarantee that all messages that arrive in order can be accepted by the destination IMP.

Suppose all single-packet messages are initially accepted (or stored). Let us take a

Table 2 Retransmission Pattern for Single-packet Messages

SOURCE IMP	DESTINATION IMP
MSG(i) sent	
MSG(i+1) sent	
MSG(i+2) sent	MSG(i+1) arr out of order
MSG(i+3) sent	MSG(i) arr
	RFNM(i) sent
	ALL(i+1) sent
	MSG(i+2) arr out of order
.	MSG(i+3) arr out of order
RFNM(i) arr	
MSG(i+4) sent	
ALL(i+1) arr	MSG(i+4) arr out of order
MSG(i+1) sent	MSG(i+1) arr
	RFNM(i+1) sent
RFNM(i+1) arr	ALL(i+2) sent
MSG(i+5) sent	
ALL(i+2) arr	MSG(i+5) arr out of order
MSG(i+2) sent	MSG(i+2) arr
	RFNM(i+2) sent
RFNM(i+2) arr	ALL(i+3) sent
MSG(i+6) sent	
ALL(i+3) arr	MSG(i+6) arr out of order
MSG(i+3) sent	MSG(i+3) arr
	RFNM(i+3) sent
RFNM(i+3) arr	ALL(i+4) sent
MSG(i+7) sent	
ALL(i+4) arr	MSG(i+7) arr out of order
MSG(i+4) sent	MSG(i+4) arr
	RFNM(i+4) sent
RFNM(i+4) arr	ALL(i+5) sent
MSG(i+8) sent	.
All(i+5) arr	.
MSG(i+5) sent	.
.	.

closer look at the situation where all single-packet messages are accepted (or stored) such that there is no reassembly buffer available for messages that have to be delivered to their HOSTs next. This is not really a lockup condition because the source IMPs keep a copy of all single-packet messages for which an RFNM has not yet been received. Therefore any single-packet message, which arrived out of order but was accepted by the destination IMP nevertheless, can be deleted later without the message being lost. The destination IMP only has to send an ALLOCATE for each deleted single-packet message to the corresponding source IMP when reassembly buffer space is available. This can also be considered as a deferred rejection. But now a retransmission is only necessary if the destination IMP is really running out of reassembly buffers.. In this case, the physical limitations of the system are reached and we cannot hope to gain large throughput increases by means of protocol changes.

V. Conclusions

In this paper we have shown that the design of flow control procedures is fraught with hidden dangers. Indeed, some of the most "obvious" principles lead to catastrophic network failures. For example, we have been able to show that both packet reassembly and message sequencing are likely sources of serious trouble even though both spring from apparently sound reasoning. Once discovered, these lockups and degradations are easily removed. However, the possibility of other, as yet undiscovered, lockups and degradations leaves one in a most uncomfortable position.

A solution to this difficulty is to subject the flow control procedure to a formal test of its "correctness". Such a procedure is discussed by Postel [12], but until the pure flow control portion of network progress can be isolated in a clean way (such as, for example, structured programming), the enormity of performing the test is prohibitive. An alternative solution would be to adopt an extremely simple approach to flow control which could be proven safe (For example, the UCLA Virtual Machine Monitor is designed around a small isolated kernel whose security can be verified due to its size [13]). Until progress is made in those directions we can look forward to the unpleasant surprises described in this paper.

Acknowledgments

We gratefully acknowledge the contributions of Bill Naylor, Joe Katz, and the remainder of the staff at UCLA-NMC.

References

[1] Roberts, L. G., and B. D. Wessler, "Computer network development to achieve resource sharing," AFIPS Conference Proc., 36, pp. 543-549, SJCC, Atlantic City, N.J., 1970.

[2] Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The interface message processor for the ARPA computer network," AFIPS Conference Proc., 36, pp. 551-567, SJCC, Atlantic City, N.J., 1970.

[3] Kleinrock, L., "Analytic and simulation methods in computer network design," AFIPS Conference Proc., 36, pp. 569-579, SJCC, Atlantic City, N.J., 1970.

[4] Frank, M., I. T. Frisch, and W. Chou, "Topological considerations in the design of the ARPA computer network," AFIPS Conference Proc., 36, pp. 581-587, SJCC, Atlantic City, N.J., 1970.

[5] Kleinrock, L., "A delay dependent queue discipline," Naval Research Logistics Quarterly, Vol. 11, No. 4, pp. 329-341, December 1964.

[6] McQuillan, T. M., W. R. Crowther, B. P. Cosell, D. C. Walden, and F. E. Heart, "Improvements in the design and performance of the ARPA network," AFIPS Conference Proc., 41, pp. 741-754, FJCC, Anaheim, California, 1972.

[7] Price, W. L., "Simulation of a packet-switched data network operating with a revised link and node protocol," NPL Report COM 68, April 1973.

[8] Davies, D. W., " The control of congestion in packet-switching networks," IEEE Transactions on Communications, COM-20, 3, pp. 546-550, June 1972.

[9] Kahn, R. E. and W. R. Crowther, "Flow control in a resource-sharing computer network," IEEE Transaction on Communications, COM-20, 3, pp. 539-546, June 1972.

[10] BBN Report No. 2717., "Interface message processors for the ARPA computer network," Quarterly Technical Report No. 4, January 1974.

[11] Crowther, W. R., J. M. McQuillan, and D. C. Walden, "Reliability issues in the ARPA Network," Third Data Communications Symposium, pp. 159-160, St. Petersburg, Florida, November 1973.

[12] Postel, J. B., "A graph model analysis of computer communications protocols," Technical Report UCLA-ENG-7410, University of California, Los Angeles, January 1974.

[13] Popek, G. J. and C. S. Kline, "Verifiable secure operating system software," AFIPS Conference Proc., 43, pp. 145-151, NCC, Chicago, Ill., 1974.

This work was supported by the Advanced Research Projects Agency of the Department of Defense (DAHC-15-73-C-0368).