#188

ARPANET LESSONS*

Leonard Kleinrock
Computer Science Department
University of California
Los Angeles, California 90024

ABSTRACT

Flow control is an essential function in computer networks but it is beset with subtle dangers. The ARPANET has taught us many lessons in this regard, some of which we discuss in this paper. Specifically, we identify and expose a number of deadlocks and degradations and then present the remedy to these traps as implemented in the ARPANET.

1. INTRODUCTION

The ARPANET experience has had an enormous impact on data communications throughout the world. Indeed the ARPANET was not only a breakthrough in its demonstration of the effectiveness of packet switching for data communications but it was also unique (in the "buy now and think later" computer industry) in that considerable effort was put forth in careful analysis and measurement during its growth and development. And yet, in spite of this effort, we were surprised at the occurrence of a number of deadlocks, degradations and traps which lay waiting for us as we delved deeper into the network design. It is our purpose in this paper to discuss these events, to collect them in one place, and to draw what lessons we can from the ARPANET experience.

By its very nature this type of paper is rather easy to write; it is always simple to discuss weak points of a system's operation after they manifest themselves. Indeed we will dwell on the shortcomings rather than on the strong points of the ARPANET. The reader is cautioned not to take this as criticism of the ARPANET, but rather as a constructive attempt to emphasize what can be learned from an experiment such as this. Without such honest evaluation and documentation (which has been the hallmark of the ARPANET development) we would have been unable to achieve the mature design technology and system development which we believe the ARPANET has provided. Indeed the ARPANET experiment has been an enormous success; it functions very well as a sophisticated communications service for the majority of its users for the majority of the time.

It is appropriate in this introduction to provide a thumbnail sketch of the background which led to the creation of the ARPANET. In the early 1960's Lawrence G. Roberts participated in an experiment whereby a computer at System Development Corporation in Santa Monica, California was accessed over a leased line from Lincoln Laboratory in Lexington, Massachusetts. Larry was so motivated by that experiment that when he came to the Advanced Research Projects Agency (ARPA) Information Processing Techniques group some few years later, he initiated the development of a modern data communications network using a technique which was later to be known as "packet switching". In 1967 he gathered roughly a dozen ARPA contractors and discussed the concept of a packet switching network with us. Our task was to create a specification that would then become a Request For Proposal which would then go out for competitive bid to industry. This network was to provide a means for sharing the computer resources of the many ARPA contractors and interested government agencies. We were keenly aware that, in order to provide an acceptable service, the network itself should demand very little from existing computer facilities and their users; our design reflected this in our attempt to connect to these systems in a non-interfering way. Among those gathered at this meeting was Professor Mel Pirtle of Berkeley who had the foresight to bang his fist on the table and insist that if the network could not provide an end-to-end response time of less than one-half second for short character strings, then it would be unable to support remote interactive use of its resources! We agreed, and so it is - the ARPANET is designed to respond to short messages in less than two-tenths of a second, and our measurements indicate that it easily beats this goal and provides extremely effective interactive use in a friendly online environment. In all major respects, however, the driving force behind the ARPANET was clearly Larry Roberts and to him goes the major credit for its development. The final specification was created early in 1968, the Request For Proposal then went out and bids were received. Early in 1969 the contract was awarded to Bolt, Beranek and Newman (BBN) and in September of 1969 the first switching computer (now well-known as the IMP) was installed at UCLA to form a one-node network! The rest is well-documented history bringing us to the present configuration which is approximately a 55-node network with three satellite channels spanning one and one-half oceans. The reader is referred to [5] for more details regarding the growth of the ARPANET, its functional description, its design technology, and its analytical and measured performance.

A computer network may be thought of as a collection of resources (switches, circuits, and processing facilities and programs), a flow of data and messages through the network, and a set of operating rules managing that flow through the hardware elements. We expect unusual user behavior and shall not dwell further on that subject [2]. However physical failures and software surprises cause problems which require special attention. For example, a hardware failure can (and did once) cause an IMP to claim it was some other IMP; havoc ensued momentarily. In another instance one IMP claimed it had zero-delay paths to all other IMPs thereby attracting (and absorbing!) vast quantities of network traffic; again this was a (memory) hardware failure. To remedy such situations one merely makes the hardware reliable and provides some safeguards against such failures. Of more interest are the software surprises, and it is these which form the subject of this paper. The culprit, in most cases, turns out to be the flow control procedure. Much of what we say has been published in bits and pieces previously [1, 3, 7]; this paper summarizes a number of lessons we have learned from the ARPANET experience (the reader is once again referred to [5] for a far more complete treatment of these and many other issues).

2. PHILOSOPHY OF FLOW CONTROL

It is not our purpose here to describe the details of the ARPANET flow control procedure. Rather we wish

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC 15-73-C-0368.

to address some of those issues which are inherent to flow control and which are capable of causing problems if not handled properly. Basically, a flow control procedure is one which judiciously allocates finite-capacity resources in a communications interchange. Its purpose is to prevent congestion while at the same time maintaining an efficient movement of traffic. If a flow control procedure is not properly designed, then rather than streamlining the flow, it may lead to lock-up or deadlock conditions, which are among the most serious system malfunctions that can occur in a computer system or computer network. The ARPANET flow control has gone through three versions in an attempt to provide efficient and safe data flow. In the following two sections, we will describe only those details of each version sufficient to characterize the lesson involved in each.

Efficient resource sharing is the goal! The finite-capacity resources we have in mind include: buffer storage capacity for packets in transit as well as at the source and destination nodes; communication channel capacity; switch processing capacity; and storage space in various system tables. This allocation of resources must be sensitive to the dynamics of the system and to the user needs. In addition, the ARPANET guarantees that the order in which data enters the net during a process-to-process communication will be the same order in which that data leaves the network, thereby guaranteeing an orderly sequencing of packets and messages at the boundaries of the network. Whereas some disagreement exists regarding whether all process-to-process communication should be restricted to sequential flow, it certainly is reasonable that a network should be in a position to provide such a function if requested. This innocent function of sequencing brings with it the potential for deadlock! In fact, any constraint on delivery of data is a potential deadlock since if the constraint (e.g., proper sequencing) cannot be met, then the flow ceases, thereby deadlocking that flow and possibly deadlocking other flows due to system resources which have been captured and which may not be releasable by that flow. Another reasonable guarantee from a network is the delivery of all traffic which is accepted (i.e.,no lost messages); similarly, one ordinarily requires that no duplicate messages be delivered.* In a sense we are on the horns of a dilemma naturally being led into constrained flow for prudent reasons and then being dragged down into quagmired flow due to lockups and serious degradations if we are not careful. The ARPANET has served us well in identifying a class of flow control problems and in finding and implementing solutions to many of them.

In the following section we describe some of the deadlock situations we have encountered along with their solutions and in Section 4 we discuss degradations (as opposed to deadlocks) to data flow.

3. DEADLOCKS AND THEIR REMEDIES

A deadlock (also known as a lockup) is a condition in which processes competing for resources cannot proceed due to an improper allocation of these resources among these processes. A simple example is the case of two processes, say P_1 and P_2, competing for two resources, say A and B. Assume that P_1 has been assigned resource A and that P_2 has been assigned B. Further assume that neither process can proceed if it is not allocated both resources A and B simultaneously. It is clear in this situation that neither process P_1 nor P_2 can proceed and this deadlock will persist forever unless some supervisory action is taken to remedy the situation. Most deadlocks are no more complicated than this example; however, these simple conditions are usually not easily recognized when they are deeply embedded in a sophisticated flow control procedure. However, it is fortunate that once these logical flaws are uncovered, they are easily removed.

In this section we shall discuss four ARPANET deadlocks which have come to be known as: reassembly deadlock; store-and-forward deadlock; the Christmas lockup; and piggyback lockup.

Reassembly lockup, the most famous of the ARPANET deadlock conditions, was due to a logical flaw in the original (version 1) flow control procedure. In the ARPANET, a string of bits to be passed through the network is broken into "messages" which are at most approximately 8000 bits in length. These messages are themselves broken into packets which are at most approximately 1000 bits in length. A message requiring more than one packet (up to a maximum of eight) is termed a multipacket message and each of these packets traverses the network independently; upon receipt at the destination node, these packets are "reassembled" into their original order and the message itself is recomposed at which time it is ready for delivery out of the network. (A more complete description of the ARPANET protocols may be found in [5].) Reassembly lockup occurred when partially reassembled messages could not be completely reassembled since the network through which the remaining packets had to traverse was congested and this prevented these packets from reaching the destination; that is, each of the destination's neighbors had given all of their relay (store-and-forward) buffers to additional packets (from messages other than those being reassembled) heading for that same destination and for which there were no unassigned reassembly buffers available. Thus the destination was surrounded by a barrier of blocked IMPs which themselves could provide no store-and-forward buffers for the needed outstanding packets and which at the same time were prevented from releasing any of their store-and-forward buffers since the destination itself refused to accept these packets due to a lack of reassembly buffers at the destination. The deadlock was simply that the remaining packets could not reach the destination and complete the reassembly until some store-and-forward buffers became free, and the store-and-forward buffers could not be released until the remaining packets reached the destination. The importance of this reassembly lockup problem indicated the need for a new flow control procedure within the ARPANET (version 2) which reserved reassembly space for any message before it was launched into the network. This reservation procedure guaranteed that the necessary resources (i.e., reassembly buffer space) would be available for all arriving packets thereby preventing the possibility of this lockup situation.

Store-and-forward lockup is another example of a lockup that can occur in a packet-switched network if no proper precautions are taken [1]. The case of "direct" store-and-forward lockup can occur under the following conditions. Let us assume that all store-and-forward buffers in some IMP A are filled with packets headed toward some destination IMP C through a neighboring IMP B and that all store-and-forward buffers in IMP B are filled with packets headed toward some destination IMP D through IMP A. Since there is no store-and-forward buffer space available in either IMP A or B, no packet can be successfully transmitted between these two IMPs and a deadlock situation results. The lesson here is to make sure that not all of the store-and-forward buffers can reside on a single output queue and

* In fact, however, one can tag a traffic stream as one which demands neither orderly delivery nor complete delivery of all the offered traffic. See below.

this is implemented in the ARPANET by restricting the maximum length of any single output queue to be less than the total collection of pooled store-and-forward buffers. "Indirect" store-and-forward lockup can occur when all the store-and-forward buffers in a loop of IMPs become filled with packets all of which travel in the same direction (clockwise or counter-clockwise), and none of which are within one hop of their destination. Both store-and-forward lockup conditions are remedied if, as in the ARPANET, more than one path exists between all pairs of communicating IMPs.

In December, 1973, the dormant Christmas lockup condition was brought to life. This lockup was exposed by collecting measurement messages at UCLA from all IMPs simultaneously. The Christmas lockup occurred when these measurement messages arrived at the UCLA IMP for which reassembly storage had been allocated but for which no reassembly blocks had been given (A reassembly block is a piece of storage used in the actual process of reassembling packets back into messages.) These messages had no way to locate their allocated buffers since the pointer to an allocated buffer is part of the reassembly block; as a consequence, allocated buffers could never be used and could never be freed! The difficulty was caused by the system first allocating buffers before it was assured that a reassembly block was available. To avoid this kind of lockup, reassembly blocks are now allocated along with the reassembly buffers for each multipacket message in the ARPANET.

Piggyback lockup is a deadlock condition which was identified by examining the flow control code and has, as far as we know, never occurred. This lockup condition comes about due to an unfortunate combination of intuitively reasonable goals implemented in the flow control procedure. One of these goals, which we have already mentioned, is to deliver messages to a destination in the same order that the source received them; the source and destination in this particular case refers to the source IMP and destination IMP, respectively. The other innocent condition has to do with the reservation of reassembly storage space at the destination IMP which we have also discussed above. In order to make this reservation procedure efficient, it is reasonable that only the first multipacket message of a long file transfer be required to make the reservation and therefore version 2 of the ARPANET flow control procedure maintained that reservation for a given file transfer as long as successive multipacket messages from that file were promptly received in succession at the source IMP. We have now laid the groundwork for piggyback lockup! Assume that there is a maximum of eight reassembly buffers in each IMP; the choice of eight is for simplicity, but the argument works for any value. Let IMP A continually transmit eight-packet messages (from some long file) to some destination IMP B such that all eight reassembly buffers in IMP B are used up by this transmission of multipacket messages. If now, in the stream of eight-packet messages, IMP A sends a single packet message (not part of the file transfer) to destination IMP B, it will generally not be accepted 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 when one of the eight-packet messages is being transmitted to its HOST). The single-packet message will therefore be treated as a request for buffer allocation (these requests are the mechanism by which reservations are made). This request will not be serviced before the RFNM (an end-to-end acknowledgement from destination to source) for the previous multipacket message has been sent. At this time, however, all the free reassembly buffers will immediately be allocated to the next multipacket message in the file transfer for efficient transmission as mentioned above; this allocation is said to be "piggybacked" on the RFNM. In this case, the eight-packet message from IMP A that arrives later at IMP B (and which is stored in the eight buffers) cannot be delivered to its destination HOST because it is out of order. The single-packet message that should be delivered next, however, will never reach the destination IMP since there is no reassembly space available and therefore its requested reservation can never be serviced. Deadlock! A minor modification removes the piggyback lockup as follows. The described deadlock can only occur because single and multipacket 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 which is due to message sequencing cannot occur.

These various deadlock conditions are usually quite easy to prevent once they are detected and understood. The trick, however, is to expurgate all deadlocks from the control mechanism ahead of time, either by careful programming (a difficult task) or by some automatic checking procedure (which may be as difficult as proving the correctness of programs). On the other hand, the deadlocks we have found in the ARPANET have been eliminated, and, in so doing, we have come to understand some of the dangers in flow control which must be avoided.

4. DEGRADATIONS AND THEIR REMEDIES

A degradation is just that, namely, a reduction in the network's level of performance. For our purposes, we shall measure performance in terms of delay and throughput. In the next four paragraphs, we discuss four sources of ARPANET degradation and their remedies, namely: looping and hold-down in the routing procedure; gaps in transmission; single-packet turbulance; and phasing.

An efficient message routing procedure is an essential ingredient for the successful operation of a computer network. The function of a routing procedure is to direct message traffic along paths within the network in a fashion which avoids congestion. As opposed to the flow control procedure which regulates how much traffic enters the network, the routing procedure must be ready to handle all traffic which the network accepts. The ARPANET uses an adaptive routing procedure which estimates delays in the network and routes traffic according to these estimates. The principal feature of this procedure is that it employs a distributed control algorithm whereby no overall decision-making authority is vested in some particular location. Rather, all nodes make local routing decisions in a dynamic fashion. Sepcifically, each node (say node n) keeps a routing table, which is simply a directory with one entry per destination in the net. This entry (say, for destination k) gives the name of the node to which node n will route all traffic it receives which is destined for node k. The entries in these tables are arrived at by the exchange of estimates among neighboring nodes in the network. A given node will route traffic to that neighbor which it estimates will provide the shortest delay in reaching the final destination. This neighbor-to-neighbor updating is really carrying global routing information. One of the successes of the ARPANET has been to demonstrate that such a distributed routing control procedure is basically stable and can converge to fairly efficient routes. It is reasonably responsive to network nodal and channel failures, but more important, it can automatically become aware of a new node as soon as it is connected (or repaired and returned) into the network. There is a cost for this adaptive routing as discussed in [4] and this includes both the

overhead in maintaining the tables as well as the interference the updating causes to data traffic. It is still not clear whether adaptive routing really pays off or whether a fixed (or deterministic alternate) routing procedure might not be more efficient [8, 9]; such a procedure would of course be required to invoke some special mechanism in the event of a link or nodal failure and/or extreme noise or congestion somewhere in the network. Indeed at any instant, an adaptive routing procedure such as described above, appears as a fixed routing procedure in the sense that between any given source-destination IMP-pair, at most one path exists along which the routing procedure permits the flow of traffic. In fact, this brings us to a rather annoying feature of distributed control adaptive routing procedures, namely, that no path at all may exist between a source and destination IMP at certain times because of looping in the routing procedure. Looping comes about due to independent decisions made by separate nodes which cause traffic to return to a previously visited node. Of course, as in the ARPANET, any reasonable adaptive routing procedure will detect these loops (through the build-up of queues and delays perhaps) and will then break the loop and guide the traffic on to its destination. However, the occurrence of loops does cause occasional large delays in the traffic flow and in some applications this is quite unacceptable. It is ironic that a remedy which was introduced to reduce the occurrence of loops, in fact made them worse in the sense that whereas they occurred less frequently, when they did occur, they persisted for a longer time (some small number of seconds). This remedy, known as "hold-down", was such that a node could not update its routing table if the change in the delay estimate sent to it by a neighbor node exceeded some minimum threshold; the philosophy here was that if some catastrophic event had taken place to cause a threshold violation, then perhaps it would be best to defer decisions until many of the neighbors in the region of that catastrophe became aware of its occurrence. As shown in [6], it was then possible to cause a loop to be formed which itself was "held-down"! Once recognized, these hold-down loops can be prevented. A strictly loop-free routing algorithm is presented in [6].

The next degradation we wish to discuss is the occurrence of gaps in the flow of traffic between a source and destination. These gaps occur because a source-destination pair (whether it be process-to-process as in version 1, source-IMP-to-destination-IMP as in version 2, or HOST-to-HOST as in version 3) is limited in the number of messages in transit which the network will allow. In version 3 the network permits up to 8 messages in flight between any HOST-HOST pair. In addition to a message number, each multipacket message must acquire two other network resources before its transmission may begin; the first of these is a reservation of reassembly space at the destination IMP (the reservation is identified by an allocate control message which is referred to as an ALL) and the second resource is an entry in what is known as the pending leader table (PLT) which, among other things, stores data which is used in constructing the header for each packet of a multipacket message. Once a message has acquired these three resources, it may then proceed through the network. Any of the three may become limiting resources as we discuss here and as we shall further discuss with regard to phasing. Let us focus on the message number limitation. Assume that the network will permit n messages in flight at a time. If n messages are in flight, then the next one may not proceed until a RFNM is returned back at the source IMP for any one of the n outstanding messages. We now observe that if the round-trip delay (i.e., the time required to send a message across the network in the forward direction and to return its RFNM in the reverse direction) is greater than the time it takes to feed the n messages into the network, then the source will be blocked awaiting RFNMs to release further messages. This clearly will introduce gaps in the message flow resulting in a reduced throughput. In a network as large as the ARPANET, such delays do exist and, in fact, it has been observed in recent measurements that when the path length exceeds 5 hops, then gaps begin to develop. Of course the remedy for this is to increase n (which we may be unwilling to do for flow control considerations).

We now come to the issue of single packet turbulance. The ARPANET was originally designed to handle only two kinds of traffic: interactive traffic characterized by short, bursty transmissions that require a small network delay; and file transfers, which generally are characterized by long sequences of multipacket messages that require a large network throughput. Both of these traffic types require high reliability in the sense that no data should be discarded by the network nor should the network deliver duplicates of the same data, and further, proper sequencing of the traffic is required. However, a third type, real-time (stream) traffic, has recently been recognized as a potential candidate for transmission through a packet-switched network. The throughput and delay requirements for real-time traffic are quite different from the throughput and delay requirements for interactive use or file transfers. For the transmission of digitized speech, for example, it is necessary to achieve a relatively high throughput with small delay for small messages since long messages result in long source delays (unacceptable for speech). Real-time traffic is not nearly as demanding in terms of reliability, however, and particularly in the case of digitized speech, we do permit packets to be discarded by the network if they arrive out of order. Indeed if we refer to Figure 1 we may characterize these three traffic types on a very simple diagram which is due to Dan Cohen of the Information Sciences Institute in Los Angeles. In that figure, we show three sides of a triangle, each one of which represents a desirable network property, namely, high reliability, large throughput, and low delay.

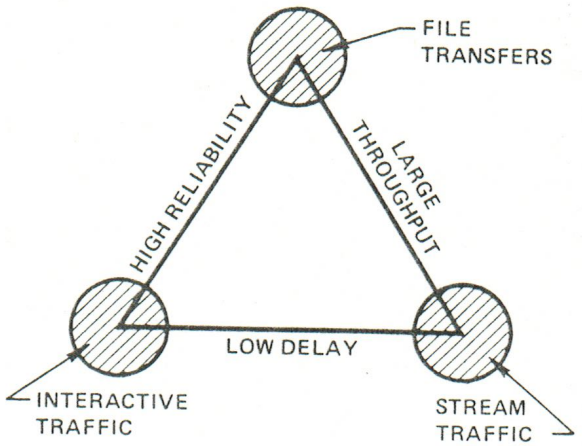

Fig. 1. Network Properties and Traffic Types

The closer one is to a given side the more the property associated with that side is realized. If, for example, one requires low delay, then one must lie anywhere along the base of the triangle; the diagram indicates that one may then also achieve either one of the two other network properties but not both. For example, if one requires high reliability along with low delay, then one moves to the lower left-hand corner of the triangle which is exactly the requirement put forward by

interactive traffic; in particular, we see that interactive traffic will not achieve a large throughput (nor does it require it ordinarily). If one chooses to locate at the apex of the triangle then one is asking for a large throughput along with a high reliability which is the typical requirement of a file transfer, and in this case we are willing to give up the low delay and accept a large initial delay in that transfer. Stream or real-time traffic finds itself in the lower right-hand corner of the triangle requiring low delay and large throughput but is willing to accept a low reliability as mentioned above. We realize that originally little effort was put forth to optimize the transmission of stream traffic in the ARPANET. It was nevertheless surprising to find that the observed throughput for single-packet stream messages was in many cases only about 1/4 of what one would expect. To understand this, recall that single-packet messages are not accepted by the destination IMP if they arrive out of order. Rather, they are then treated as a request for the allocation of one reassembly buffer. Therefore if, in a stream of single-packet messages, packet p arrives out-of-order, (say it arrives after packet p+3), then packets p+1, p+2 and p+3 will all be discarded at the destination IMP and only after packet p arrives will a single packet buffer be allocated to message p+1. This allocation will piggyback on the RFNM for packet p and when it arrives at the source IMP, it will then cause a retransmission of the discarded packet p+1 (which has been stored in the source IMP). Of course any packets arriving at the destination after packet p+3, but before p+1 arrives in order, will themselves be discarded. When packet p+1 finally arrives for the second time at the destination IMP it is now in order and this will cause an allocation of a single-packet buffer to packet p+2, etc. The net result is that only one packet will be deliverable to the destination per round-trip time along this path; had no packets been received out-of-order, then we would have been pumping at a rate close to n packets per round-trip time (if the maximum number in transit, n, could fit into the pipe). Observe that once a single packet arrives out-of-order in this stream, then the degradation from n to 1 packet per round-trip time will persist forever until either some supervisory action is taken or until the traffic stream ceases and begins again from a fresh start in the future. We refer to this effect as "single packet turbulance" and it was observed in the ARPANET as described in [7]. To remedy this single packet turbulance for stream traffic, the ARPANET has introduced a new type of message which in fact has no message number limitation, does not require orderly delivery and does not require complete delivery of all the traffic, i.e., out-of-order packets may be discarded; with the relaxation of these requirements, single packet turbulance immediately disappeared and now we are capable of supporting packetized speech through the ARPANET.

The last degradation we wish to discuss is known as "phasing". Phasing, a cause for throughput degradation, is due to the sending of superfluous requests for the reservation of reassembly space at the destination IMP; these requests are called REQALLs. A REQALL is called superfluous if it is sent while a previous REQALL is still outstanding. This situation can arise if message i sends a REQALL but does not use the ALL returned by this REQALL because it obtained its reassembly buffer allocation piggybacked on a RFNM for an earlier message (which reached the source IMP before its requested ALL). The sending of superfluous REQALLs is undesirable because it unnecessarily uses up resources. Recall that each multipacket message requires three resources to proceed and these are the message number, an ALL, and a PLT entry. The problem with phasing is that the three kinds of resources just mentioned do not all appear at the source IMP at the same time and therefore an incoming message must wait until it collects one of each kind of resource before proceeding. For example, in version 3 of the flow control procedure, there are at most 8 message numbers between a source and destination HOST, at most 4 ALL's available from the destination IMP, and at most 6 PLT entries at a source IMP. One would naturally conjecture that the limiting resource is the ALL; this is not always correct! A stable configuration (and one that was observed through measurement) is to have two messages in transit between a source and destination IMP (each of which contains a message number, a PLT and an ALL), to have one RFNM on the way back from the destination to the source IMP (which also contains these 3 resources) and to have 3 RFNMs waiting in the destination IMP to be returned to the source IMP as shown in Figure 2.

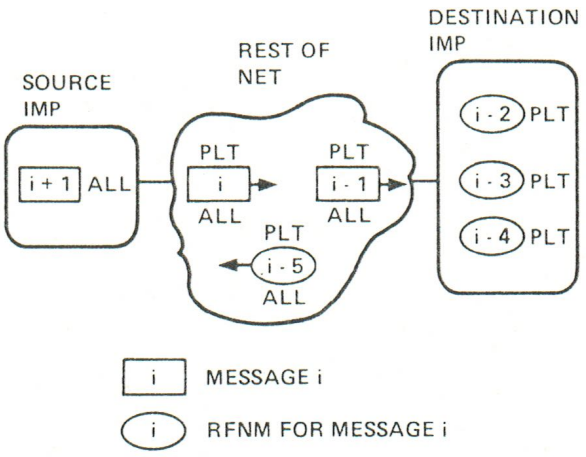

Fig. 2. The Effect of Phasing

Each of the RFNMs owns a message number and a PLT entry (but has already released its ALL); the rub is that a RFNM prefers not to be returned to a source IMP unless it also piggybacks an ALL. From Figure 2 we see that the RFNM for message i-4 will be launched in the reverse direction as soon as message i-1 arrives and releases its ALL. The critical point is that message i+1 is waiting in the source IMP with two of its three resources already acquired (namely, its message number and its ALL) but has yet to receive its third resource (a PLT). Although only 4 ALLs are available, we see that 7 message numbers have been assigned, 4 of which are attached to RFNMs and only 3 of which are attached to messages. It is ironic that the ALL is not the limitation at the source node, but rather the abundant PLT's turn out to be the limiting resource in this case since so many of them are piled up in the destination IMP at the wrong end of the network! A similar (and worse effect) occurs when fewer ALLs are permitted (see [3]). There are two obvious ways to avoid this undesirable phasing of messages. First, one can avoid sending superfluous REQALLs that are the underlying cause for the phasing. Second, one can avoid the piggybacking of allocates on RFNMs as long as there are other replies to be sent.

In this section we have succinctly described some of the known degradations to the two performance measures (delay and throughput) as observed and remedied in the ARPANET.

5. CONCLUSIONS

In this paper we have seen that a necessary function for a computer network is to control the flow of traffic at its entry points. This flow control not only throttles traffic but also guarantees some end-to-

end responsibility in the form of proper ordering of messages, detection and deletion of duplicates, and prevention of lost messages. We have seen that these very reasonable goals are capable of leading to deadlocks and degradations in network performance. In fact we have seen that any constraint put on the flow of traffic is capable of producing deadlocks and degradations since if the constraint cannot be met then a deadlock will occur, whereas if the constraint is slow in being met then a degradation will result. To avoid these problems, one is forced to simplify the flow control functions as well as their implementation as far as possible and, beyond that, to be on constant alert for possible difficulties, both through measurement and observation. Once a deadlock or degradation is discovered we have seen how simple it is to remedy, as we have done in the ARPANET. At this point in the development of networks it is not clear to this author how one can guarantee the absence of deadlocks and degradations through a reliable and efficient test. Hopefully, cleaner code and cleaner concepts of flow control will enhance the ability to conduct such tests in the future.

REFERENCES

[1] Kahn, R.E. and W.R. Crowther, "Flow Control in a Resource Sharing Computer Network," Proceedings of the Second IEEE Symposium on Problems in the Optimization of Data Communications Systems, Palo Alto, California, 108-116, October 1971.

[2] Kleinrock, L. and W.E. Naylor, "On Measured Behavior of the ARPA Network," AFIPS Conference Proceedings, 1974 National Computer Conference, Vol.43, 767-780, 1974.

[3] Kleinrock, L. and H. Opderbeck, "Throughput in the ARPANET - Protocols and Measurement," Proceedings of the Fourth Data Communications Symposium, Quebec, Canada, 6-1 to 6-11,October 1975.

[4] Kleinrock, L., W.E. Naylor, and H. Opderbeck, "A Study of Line Overhead in the ARPANET," Communications of the Association for Computing Machinery, Vol. 19, 3-13, January 1976.

[5] Kleinrock, L., Queueing Systems, Vol. 2: Computer Applications, Wiley Interscience (New York), 1976.

[6] Naylor, W.E., "A Loop-Free Adaptive Routing Algorithm for Packet Switched Networks," Proceedings of the Fourth Data Communications Symposium, Quebec, Canada, 7-9 to 7-14, October 1975.

[7] Opderbeck, H. and L. Kleinrock, "The Influence of Control Procedures on the Performance of Packet-Switched Networks," Proceedings of the National Telecommunications Conference, San Diego, California, December 1974.

[8] Price, W.L., "Further Simulation Experiments on Adaptive Routing Using Locally Available Parameters," U.K. National Physical Laboratory, Division of Computer Science, NPL Report COM 81, December 1975.

[9] Rudin, H., "On Routing and "Delta-Routing": Techniques for Packet-Switched Networks," Proceedings of the IEEE International Conference on Communications, San Francisco, California, 41-20 to 41-24, June 16-18, 1975.

IEEE CATALOG NUMBER
76 CH 1085-0 CSCB

CONFERENCE RECORD

1976 INTERNATIONAL CONFERENCE ON COMMUNICATIONS

Volume II

Communications, Cornerstone of Freedom

ICC76 • JUNE 14-16
PHILADELPHIA
PENNSYLVANIA