Flow Control: A Comparative Survey

MARIO GERLA, MEMBER, IEEE, AND LEONARD KLEINROCK, FELLOW, IEEE

(Invited Paper)

Abstract—**Packet switching offers attractive advantages over the more conventional circuit-switched scheme, namely, flexibility in setting up user connections and more efficient use of resources after the connection is established. However, if user demands are allowed to exceed the system capacity, unpleasant congestion effects occur which rapidly neutralize the delay and efficiency advantages. Congestion can be eliminated by using an appropriate set of traffic monitoring and control procedures called flow control procedures. Flow control can be exercised at various levels in a packet network. The following levels are identified and discussed in this paper: hop level, entry-to-exit level, network access level, and transport level. For each level, the most representative techniques are surveyed and compared. Furthermore, the interaction between the different levels is discussed.**

I. INTRODUCTION

A packet-switched network may be thought of as a distributed pool of productive resources (channels, buffers, and switching processors) whose capacity must be shared dynamically by a community of competing users (or, more generally, processes) wishing to communicate with each other. Dynamic resource sharing is what distinguishes packet switching from the more traditional circuit switching approach, in which network resources are dedicated to each user for an entire session. The key advantages of dynamic sharing are greater speed and flexibility in setting up user connections across the network and more efficient use of network resources after the connection is established.

These advantages of dynamic sharing do not come without a certain danger, however. Indeed, unless careful control is exercised on the user demands, the users may seriously abuse the network. In fact, if the demands are allowed to exceed the system capacity, highly unpleasant congestion effects occur which rapidly neutralize the delay and efficiency advantages of a packet network. The type of congestion that occurs in an overloaded packet network is not unlike that observed in a highway network. During peak hours, the demands often exceed the highway capacity, creating large backlogs. Furthermore, the interference between transit traffic on the highway and on-ramp and off-ramp traffic reduces the effective throughput of the highway, thus causing an even more rapid increase in the backlog. If this positive feedback situation persists, traffic on the highway may come to a standstill. The typical relationship between effective throughput and offered load in a highway system (and, more generally, in many uncontrolled, distributed dynamic sharing systems) is shown in Fig. 1.

By properly monitoring and controlling the offered load many of these congestion problems may be eliminated. In a highway system, it is common to control the input by using access ramp traffic lights. The objective is to keep the interference between transit traffic and incoming traffic within acceptable limits, and to prevent the incoming traffic rate from exceeding the highway capacity.

Similar types of controls are used in packet switched networks, and are called *flow control* procedures. As in the highway system, the basic principle is to keep the excess load out of the network. The techniques, however, are much more sophisticated since the elements of the network (i.e., the switching processors) are intelligent, can communicate with each other, and therefore can coordinate their actions in a distributed control strategy.

Internal network congestion may also be relieved by rerouting some of the traffic from heavily loaded paths to underutilized paths. It is important to understand, however, that *routing* can reduce and, perhaps, delay network congestion; it cannot prevent it. We do not discuss the interactions between routing and flow control in this paper. The interested reader is referred to the routing protocol survey paper by Schwartz and Stern in this TRANSACTIONS [51].

The main functions of flow control in a packet network are:

1) prevention of throughput degradation and loss of efficiency due to overload,

2) deadlock avoidance,

3) fair allocation of resources among competing users, and

4) speed matching between the network and its attached users.

Throughput degradation and deadlocks occur because the traffic that has already been accepted into the network (i.e., traffic that has already been allocated network resources) exceeds the nominal capacity of the network. To prevent overallocation of resources, the flow control procedure includes a set of constraints (on buffers that can be allocated, on outstanding packets, on transmission rates, etc.) which can effectively limit the access of traffic into the network or, more precisely, to selected sections of the network. These constraints may be fixed, or may be dynamically adjusted based on traffic conditions.

Apart from the requirement of throughput efficiency, network resources must be fairly distributed among users.

Manuscript received December 15, 1979; revised Jaunary 8, 1980. This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-77-C-0272.

The authors are with the Department of Computer Science, University of California, Los Angeles, CA 90024.

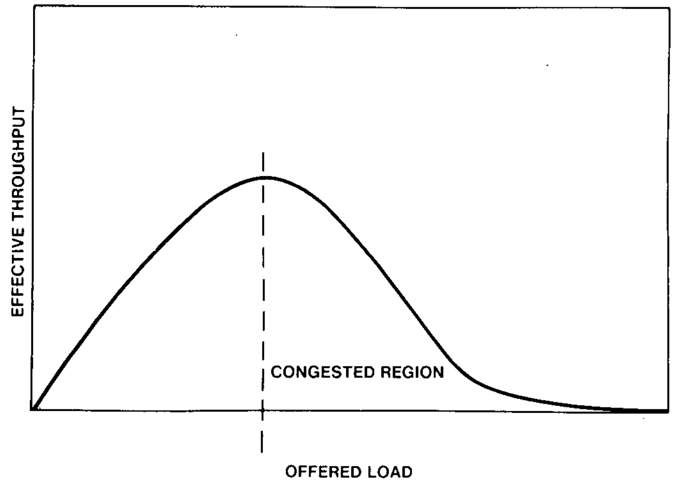


Fig. 1. Effective throughput versus offered load in an uncontrolled, distributed dynamic sharing system.

Unfortunately, efficiency and fairness objectives do not always coincide. For example, referring back to our highway traffic situation, the effective throughput of the Long Island Expressway could be maximized by opening all the lanes to traffic from the Island to New York City during the morning rush hour, and in the opposite direction during the evening rush hour. This solution, however, would also maximize the discontent of the reverse commuters (and we all know how dangerous it is to anger a New Yorker)! In packet networks, unfairness conditions can also arise (as we will show in the following sections); but they tend to be more subtle and less obvious than in highway networks because of the complexity of the communications protocols. One of the functions of flow control, therefore, is to prevent unfairness by placing *selective* restrictions on the amount of resources that each user (or user group) may acquire, in spite of the negative effect that these restrictions may have on dynamic resource sharing and, therefore, overall throughput efficiency.

Flow control can be exercised at various levels in a packet network. The following levels are identified and discussed in this paper.

1) *Hop Level*: This level of flow control attempts to maintain a smooth flow of traffic between two neighboring nodes in a computer network, avoiding local buffer congestion and deadlocks. (We shall devote Section II to the discussion of this form of flow control.)

2) *Entry-to-Exit Level*: This level of flow control is generally implemented as a protocol between the source and destination switch, and has the purpose of preventing buffer congestion at the exit switch (Section III).

3) *Network Access Level*: The objective of this level is to throttle external inputs based on measurements of internal (as opposed to destination) network congestion (Section IV).

4) *Transport Level*: This is the level of flow control associated with the transport protocol, i.e., the protocol which provides for the reliable delivery of packets on the "virtual" connection between two remote processes. Its main purpose is to prevent congestion of user buffers at the process level (i.e., outside of the network) (Section V).

Some authors reserve the term "flow control" for the transport level, and refer to the other three levels of control as *congestion control* [34]. This terminology is used to emphasize the physical distinction between the first three levels, which are realized in the communications subnet (and therefore are the responsibility of the network implementer) and the fourth level, which is realized in the user devices (and therefore is the responsiblity of the network customer). In this paper, we have chosen to use the term flow control for all four levels.

The design of an efficient flow control strategy for a packet network is a complex task in many ways. The most critical issue is the fact that flow control is a multilayer distributed protocol involving several different levels. At each level, the flow control implementation must be consistent and compatible with other protocol functions existing at the same level. Furthermore, the interactions between different levels must be carefully studied in order to avoid duplication of functions on one hand, and lack of coordination on the other.

The purpose of this paper is to provide a taxonomy of flow control mechanisms based on the above defined multilevel structure. First, we review problems, functions, and performance measures of flow control. Then, for each level we survey the most representative flow control techniques that have been proposed and/or implemented, providing a performance comparison among techniques at the same level, and discussing the interaction between techniques at different levels. Finally, we briefly mention some new flow control issues raised by novel computer network applications.

II. FLOW CONTROL: PROBLEMS, FUNCTIONS, AND MEASURES

Our overall problem is to identify mechanisms which permit efficient dynamic sharing of the pool of resources (channels, buffers, and switching processors) in a packet network. In this section, we first describe and illustrate the congestion problems caused by *lack* of control. Then we define the functions of flow control and the different levels at which these functions are implemented. Finally, we introduce performance measures for the evaluation and comparison of different flow control schemes.

A. Loss of Efficiency

The main cause of throughput degradation in a packet network is the *wastage* of resources. This may happen either because conflicting demands by two or more users make the resource unusable (e.g., collisions on a random access channel); or because a user acquires more resources than strictly needed, thus starving other users (e.g., a slow sink fed by a fast source may create a backlog of packets within the network which prevents other traffic from getting through). The two resources that are most commonly "wasted" in a packet network are *communications capacity* and *storage capacity*.

Buffer wastage is an indirect consequence of limited nodal storage: a given end-to-end packet stream may be blocked at an intermediate node along the path because all of the buffers have been "hogged" by other streams. This may happen even if channel bandwidth is plentiful along the blocked stream path, thus causing an unnecessary loss of throughput. The source of this throughput degradation is that some users unnecessarily monopolize (i.e., waste) the buffers at some congested node.

A simple example of throughput degradation caused by buffer interference is shown in Fig. 2. Two pairs of hosts, (A, A') and (B, B'), are engaged in data transmission through a single network node. Access line speeds (in arbitrary units) are given in Fig. 2. The traffic requirement from A to A' is constant and is equal to 0.8 (measured in the same units as the line speed). The requirement from B to B' is variable, and is denoted by λ. When λ approaches 1, the output queue from the switch to host B' grows indefinitely large filling up all the buffers in the switch. Packets arriving when all the buffers are full are discarded, and are later retransmitted by the source host (we refer to this model as the *retransmit model*). If we plot the total throughput, i.e., the sum of (A, A') and (B, B') delivered traffic as a function of λ (as in the solid curve of Fig. 3), we note that for $\lambda = 1$, the throughput experiences a sharp drop from 1.8 to 1.1. The drop is due to the fact that the switch can handle the entire user demand $= \lambda + 0.8$ for $\lambda < 1$; while for $\lambda \geqslant 1$, the switch buffers become full, causing overflow. Consequently, large queues build up in both the A and B hosts. With a heavy load, the rate of packet transmissions (and retransmissions) from B is 10 times the rate from A because of the difference in line access speeds. Thus, packets from B have a 10 times better chance of being accepted when a buffer becomes free than packets from A, leading to a 10 to 1 imbalance in effective throughput. Since the (B, B') throughput is limited to 1, the (A, A') throughput is reduced to 0.1 (i.e., one tenth of the AA' throughput), yielding a total throughput $= 1.1$ for $\lambda \geqslant 1$.

In this example, we have observed a *decrease in useful throughput* caused by an *increase of offered load* beyond the critical system capacity. This throughput degradation is typical of congested systems, and is often taken as a definition of congestion (i.e., a system is "congestion-prone" if an increment in offered load causes a reduction in throughput) [27].

In the previous example, we assumed that dropped packets would be retransmitted from the host. A similar analysis can be carried out assuming that dropped packets are lost (*loss model*). The throughput versus offered load performance is similar to that of the retransmit model, although the drop is somewhat smoother in this case (the dashed curve of Fig. 3).

Throughput degradation effects, caused by inefficient allocation (and therefore wastage) of buffers are found also in multinode networks as reported by several studies [27], [13], [18]. To prevent this type of degradation, proper buffer allocation rules are generally established at each node, as soon described.

Another cause of throughput degradation is channel wastage. This problem manifests itself very clearly in multiaccess channels (e.g., packet satellite, or packet radio channels), when users transmit packets at random times without prior coordination (random access). A well-known example is offered by the ALOHA channel [23]. Packets that collide are lost, thus causing channel wastage and consequently, throughput degradation. Congestion prevention in multi-

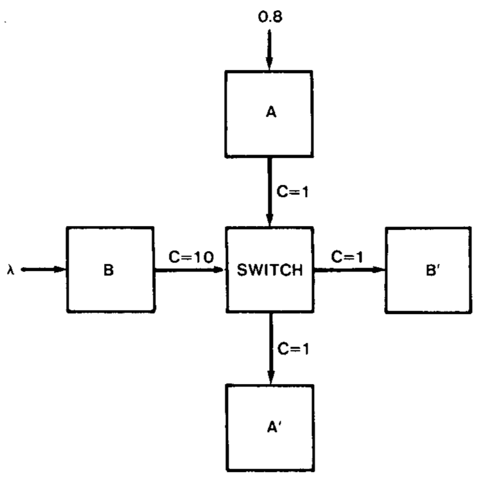


Fig. 2. Buffer interference example.

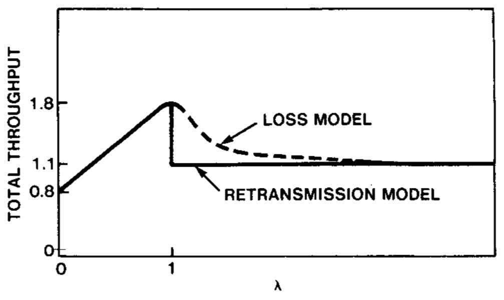


Fig. 3. Throughput degradation of system of Fig. 2 due to buffer interference.

access channels is discussed in [46] in this TRANSACTIONS. Also, it is clear that unnecessary retransmissions of a packet represent another form of channel wastage. Yet another manifestation is the use of unnecessarily long paths in a network (e.g., looping in routing algorithms).

B. Unfairness

Unfairness is a natural byproduct of uncontrolled competition. Some users, because of their relative position in the network or the particular selection of network and traffic parameters, may succeed in capturing a larger share of resources than others, and thus enjoy preferential treatment.

One example of unfairness has already been given in Figs. 2 and 3 where the $(B\text{-}B')$ flow is allowed to exceed the $(A\text{-}A')$ flow by a factor of ten. Another obvious example of unfairness is offered by the single switch loss model in Fig. 4. The speed of the output trunk is 1. Hosts A and B are injecting data into the switch with rates 0.5 and λ, respectively. For fairness, the output trunk should be equally shared by the two hosts. However, the *loss model* performance results shown in Fig. 5 indicate that for large values of λ, host B captures the entire output trunk bandwidth, reducing the A throughput to zero. As previously observed, for $\lambda \gg 0.5$ host B has a far better chance to seize free buffers in the switch than host A. Specifically, the ratio of A-packets to B-packets in the switch at heavy load is roughly equal to $0.5/\lambda$. Thus, the ratio of A-throughput to B-throughput is also $0.5/\lambda$, explaining the behavior in Fig. 5.

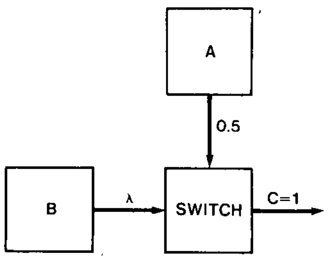


Fig. 4. Example of unfairness.

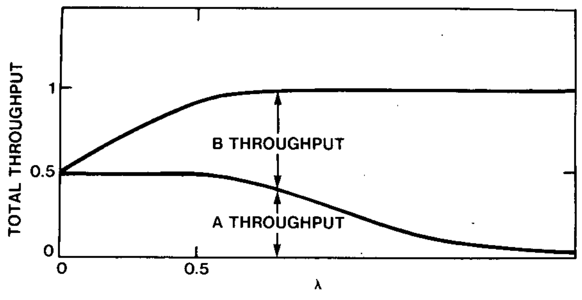


Fig. 5. Performance of system shown in Fig. 4.

Cases of unfairness have been reported in many multinode network studies, and several "fairness" techniques have been proposed. Unfortunately, the problem of fairness is considerably more difficult to deal with than the problem of total throughput degradation because a general, unambiguous definition of fairness is not always possible in a distributed resource sharing environment.

C. Deadlocks

A deadlock condition manifests itself by a (total or partial) network crash. Deadlocks often occur because of a cyclic wait of resources to become available. That is, one user is holding a portion of the resources that he currently needs and is waiting for another user to release the remaining resources necessary to complete his task and this user is waiting for yet a third user, etc., such that the sequence of "waiting" users closes into a cycle, and it is immediately seen that no user in the cycle can make any progress [3]. Thus, the throughput for this subset of users is reduced to zero.

Deadlocks are likely to occur in a network when the offered load exceeds network capacity. For a simple example of a deadlock, consider two switches, A and B, connected by a trunk carrying heavy traffic in both directions (see Fig. 6). Under the heavy traffic assumption, node A rapidly fills up with packets directed to B; and vice versa, B fills up with packets directed to A. If we assume that dropped packets are retransmitted, then each node must hold a copy of each packet (and therefore a buffer) until the packet is accepted by the other node. This may result in an endless wait in which a node holds all of its buffers to store packets being transmitted to the other node, and keeps retransmitting packets to the other node waiting for buffers to be freed there. Consequently, no useful data are transferred on the trunk. It turns out that this type of deadlock (known as direct store-and-forward deadlock [19]) is relatively easy to prevent by

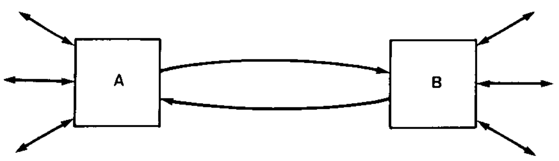


Fig. 6. Deadlock example.

setting simple restrictions on buffer usage at each node. A more extensive discussion of deadlocks will be given in Section II.

It is important to point out that buffer deadlocks are possible only in networks which retransmit dropped packets, i.e., which save a copy of a packet at each node while transmitting the packet to the next node on the path, and retransmit a copy of the packet in case of overflow (retransmit model). If dropped packets are not retransmitted (i.e., a loss model), the sending node is not required to save a copy of the packet until acceptance at the next node, thus removing a necessary condition for deadlocks. Thus, lossy networks are deadlock free; however, an additional recovery mechanism for lost packets must then be provided at the end-to-end level.

D. Flow Control Functions

Flow control may be defined as a protocol (or more generally, a set of protocols), designed to protect the network from problems related to overload and speed mismatches. Solutions to the three problems just discussed (maintaining efficiency, fairness and freedom from deadlock) are accomplished by setting rules for the allocation of buffers at each node and by properly regulating and (if necessary) blocking the flow of packets internally in the network as well as at network entry points. Actually, multiple levels of flow control are generally implemented in a network, as we shall see.

Efficiency and congestion prevention benefits of flow control do not come for free. In fact, flow control (as with any other form of control in a distributed network) may require some exchange of information between nodes to select the control strategy and possibly, some exchange of commands and parameter information to implement that strategy. This exchange translates into channel, processor, and storage overhead. Furthermore, flow control may require the dedication of resources (e.g., buffers, bandwidth) to individual users, or classes of users, thus reducing the statistical benefits of complete resource sharing. Clearly, the tradeoff between gain in efficiency (due to controls) and loss in efficiency (due to limited sharing and overhead) must be carefully considered in designing flow control strategies. This tradeoff is illustrated by the curves in Fig. 7, showing the effective throughput as a function of offered load. The ideal throughput curve corresponds to perfect control as it could be implemented by an ideal observer, with complete and instantaneous network status information. Ideal throughput follows the input and increases linearly until it reaches a horizontal asymptote corresponding to the maximum theoretical network throughput. The controlled throughput curve is a typical curve that can be obtained with an actual

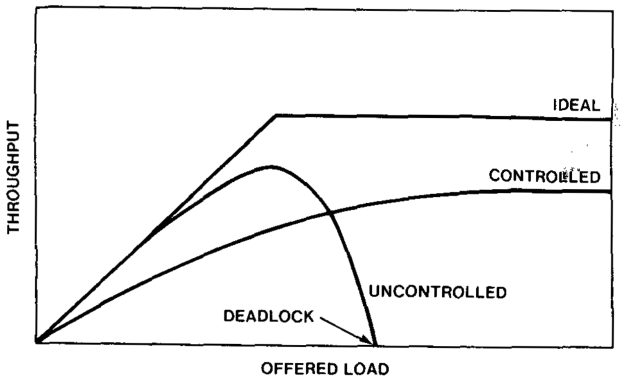


Fig. 7. Flow control performance tradeoffs.

control procedure. Throughput values are lower than with the ideal curve because of imperfect control and control overhead. The uncontrolled curve follows the ideal curve for low offered load; for higher load, it collapses to a very low value of throughput and, possibly, to a deadlock.

Clearly, controls buy safety at high offered loads at the expense of somewhat reduced efficiency. The reduction in efficiency is measured in terms of higher delays (for light load) and lower throughput (at saturation). Furthermore, experience shows that flow control procedures are quite difficult to design and ironically, can themselves be the source of deadlocks and degradations. In particular, when one controls flow, one places *constraints* on the flow. If one cannot meet a constraint, then the result is a deadlock. Or, if one is slow in meeting the constraint, the result is a throughput degradation.

E. Levels of Flow Control

Flow control in a packet network can be best described as a multilayered structure consisting of several mechanisms operating independently at different levels. Since flow control levels are closely related to (and sometimes imbedded in) protocol levels, it is helpful for us to begin by briefly reviewing the network protocol structure, pointing to the flow control provisions existing at each level [17]. The flow control level structure will then be defined following the protocol structure model.

Fig. 8 depicts the typical protocol layer architecture implemented in a packet network, using as a reference a network path connecting user devices called DTE's (data terminating equipment) through a number of intervening communications switches called DCE's (data communications equipment). For the user-to-network (i.e., DTE-to-DCE) interface, a standard set of protocol levels is now being defined by ISO and ANSI [9]. For the internode protocols within the communications subnetwork, there is less emphasis on standardization since different network manufacturers tend to select different solutions to best exploit their equipment capabilities. In spite of these differences, it is still possible to define a set of reference levels for internal network protocols which closely parallel the DTE-DCE interface protocol levels.

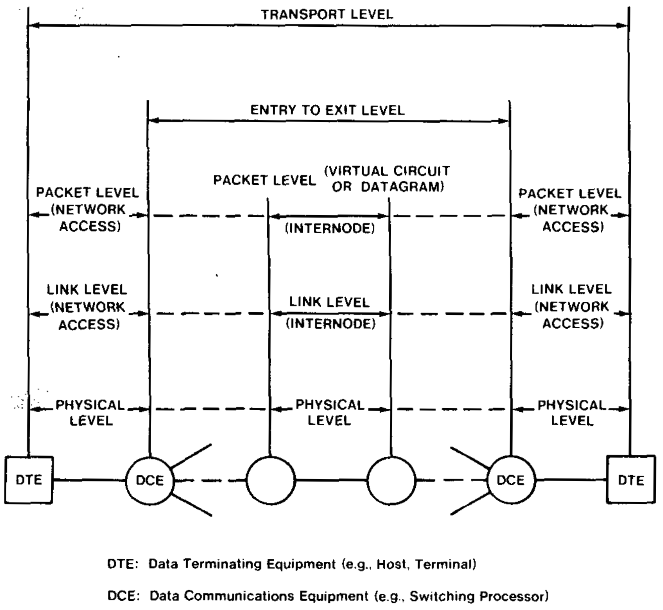

TRANSPORT LEVEL
ENTRY TO EXIT LEVEL
PACKET LEVEL (NETWORK ACCESS)
PACKET LEVEL (VIRTUAL CIRCUIT OR DATAGRAM)
(INTERNODE)
PACKET LEVEL (NETWORK ACCESS)
LINK LEVEL (NETWORK ACCESS)
LINK LEVEL (INTERNODE)
LINK LEVEL (NETWORK ACCESS)
PHYSICAL LEVEL
PHYSICAL LEVEL
PHYSICAL LEVEL
DTE
DCE
DCE
DTE
DTE: Data Terminating Equipment (e.g., Host, Terminal)
DCE: Data Communications Equipment (e.g., Switching Processor)

Fig. 8. Network protocol levels.

Starting from the bottom of the protocol hierarchy, we have the *physical level* which has the function of activating and deactivating the electrical connection between the nodes. No flow control functions are assigned to this level.

Above the physical level, we have the *link level* which serves the purpose of transporting packets reliably across individual physical links. One of the functions of this protocol is related to flow control, and consists of retransmitting packets that are dropped because of congestion at the receiving node. In some protocols, a congested receiver may stop the sender by using appropriate commands (e.g., RNR: receiver not ready, in HDLC and SDLC; or, XOFF in asynchronous terminal connections). As mentioned before, we find two different types of links in the network: the internal (or node-to-node) link and the network access link. Correspondingly, we have (at the same level in the protocol hierarchy) two types of link protocol: the *network access protocol* and the *node-to-node protocol.* Typical examples of link protocol implementation are HDLC, SDLC, and X.25 level 2 (which is a subset of HDLC).

Above the link level, we have the *packet level protocol,* which defines the procedures for establishing end-to-end user connections through the network, and specifies the format of the control information used to route packets to their destinations. Two different versions of packet protocol exist: the *virtual circuit* protocol and the *datagram* protocol.

When the virtual circuit (VC) implementation is used, a "virtual" circuit connection must be set up between a pair of users (or processes) wishing to communicate with each other *before* the data transfer can be started. The establishment of this circuit implies dedication of resources of one form or another along the network path. A typical virtual circuit implementation, used in Transpac [7], assigns a fixed path to each connection at setup time. A virtual circuit ID number, stamped in the packet header, uniquely identifies the packets belonging to a connection, and is used to route packets to the destination using routing maps implemented at each intermediate node at set up time. From the flow control point of view, the VC protocol has the distinguishing feature of permitting *selective* flow control on each individual user connection. This selective flow control can be applied at the internode level as well as at the network access level. Since a fixed path is maintained for the entire user session, the selective flow control can also be extended from entry to exit switch and if so desired, even from entry to exit DTE.

As opposed to the virtual circuit implementation, the datagram implementation does not require any circuit set-up before transmission. Each packet is independently submitted to the network, and explicitly carries in its header all the information required for its delivery to destination [34]. Selective flow control, on a connection by connection basis, is not available in the datagram implementation since the packet header does not contain specific connection information (it merely posts source and destination DTE addresses).

Above the packet protocol, we find (within the subnet only) the *entry-to-exit* (ETE) protocol. The objective of this protocol is the reliable transport of single and multipacket messages from network entry to network exit node. Important functions of this protocol which are related to flow control are the reassembly of multipacket messages at the exit node and the regulation of input traffic using buffer allocation and windowing techniques. Some network implementations do not have the ETE level of protocol. In this case, the ETE functions are relegated to higher level protocols.

The highest level of network protocol which has impact on flow control is the *transport protocol.* This protocol provides for the reliable delivery of packets on the "virtual" connection between two remote processes. One of the flow control related functions of this protocol is the protection of destination buffers. The goal is to regulate the flow so as to make the most efficient use of network resources, while avoiding buffer overflow at the destination. "Window" and "credit" schemes are generally used for this purpose.

The above network protocol review has identified various flow control functions and capabilities built into different levels of protocols, and has brought to our attention the fact that each protocol level has its own distinct flow control responsibilities. It is now clear that the classification into the four types of flow control procedures mentioned earlier parallels the classification of network protocols. Recall that there is:

1) hop (or node-to-node) level (Section II),
2) network access level (Section III),
3) entry-to-exit level (Section IV), and
4) transport level (Section V).

The diagram in Fig. 9 illustrates these levels of flow control for a typical network path. A comparison with Fig. 8 reveals the close relationship between flow control and protocol level structures.

Unfortunately, the true system behavior is far more complex than our models and classifications attempt (or can afford) to portray. Therefore, actual networks may not always mechanize all of the above four levels of flow control with distinct procedures. It is quite possible, for example, for a

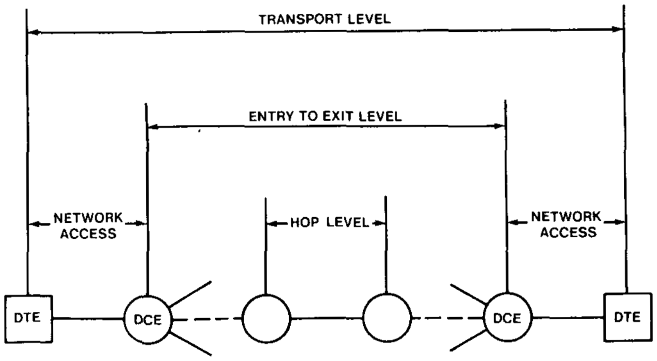


Fig. 9. Flow control levels.

single flow control mechanism to combine two or more levels of flow control. On the other hand, it is possible that one or more levels of flow control may be missing in the network implementation. The matrix in Fig. 10 provides a synopsis of the main network implementations and flow control schemes that will be surveyed in this paper. It is seen that some of the schemes cover more than one level.

F. Performance Measures

We wish to define a quantitative measure of flow control performance for various reasons. First, we wish to be able to "tune" the parameters of a given flow control scheme so as to optimize a well defined performance criterion. Second, we wish to carefully weigh performance benefits against overhead introduced by flow control. Third, we are interested in comparing the performance of alternative flow control schemes in quantitative terms.

Throughput efficiency (where throughput is expressed in packets/s) is probably the most common measure of flow control performance. Total effective throughput (sum of all the individual contributions) is evaluated as a function of offered load. This representation is particularly useful to determine the critical load in an uncontrolled system and to assess the throughput efficiency of a controlled network at heavy load.

Another common measure is the *combined delay and throughput performance.* The delay-throughput profile allows us to determine the delay overhead introduced by the controls (which the throughput versus offered load curve did not display). In general, it gives us a more complete picture of system performance than does throughput behavior alone. In fact, a system may be designed to deliver high throughput at heavy load, and yet it may experience intolerable delays at light load.

A more compact measure of combined throughput and delay performance is offered by the concept of "*power*" [13], [24]. The simplest definition of power is the ratio of throughput over delay; it is, therefore, a function of the offered load. In fact, it defines the "knee" of the throughput-delay profile as that point where power is maximized, and as shown in Fig. 11 this knee occurs where a ray out of the origin is tangent to the performance profile [24]. A very nice characterization of this maximum power point is such that it occurs when the average buffer occupancy at each

	ARPA			TRANSPAC	SNA			GMDNET	
	CQL	RFNM	NCP	X.25	SDLC	VR PACING	SESSION PACING	I-C	SBP
HOP LEVEL	●			●	●	●		●	●
NETW. ACC.				●			●	●	
ENTRY-EXIT		●		●		●	●		●
TRANSP.			●	NOT DEFINED			●	NOT DEFINED	

Fig. 10. Classification of actual flow control implementations.

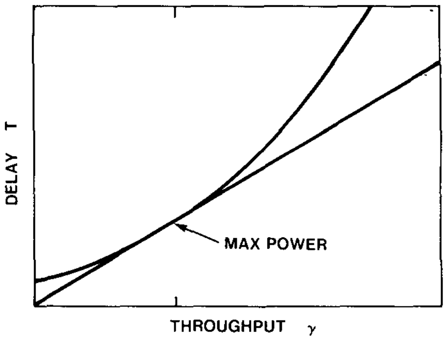


Fig. 11. Delay, throughput, and power.

intermediate node on the path is unity. In [25], it was shown that blocking due to loss systems could easily be included in a more general definition of power (by multiplying the simple definition by one minus the blocking probability); this leads to system designs whose optimum operating point is easily found and which corresponds to the operating point one would intuitively choose. Much more general definitions of power are also studied in [25].

In some important cases, power is maximized for a value of offered load which is approximately half of the saturation load [24]. The maximum power value reflects both delay performance (at light load) and throughput performance (at heavy load) and therefore, represents a good figure of merit of the flow control implementation.

III. HOP LEVEL FLOW CONTROL

A. Objective

The objective of hop level flow control is to prevent store-and-forward buffer congestion and its consequences, namely, throughput degradation and deadlocks. Hop level flow control operates in a local, "myopic" way in that it monitors local queues and buffer occupancies at each node and rejects store-and-forward (S/F) traffic arriving at the node when some predefined thresholds (e.g., maximum queue limits) are exceeded. The function of checking buffer thresholds and discarding (and later retransmitting) packets on a network link is often carried out by the data link control protocol.

This locality of the control does not preclude, however, possible end-to-end repercussions of hop level flow control due to the "backpressure" effect [i.e., the propagation of buffer threshold conditions from the congested node upstream to the traffic source(s)]. In fact, the backpressure property is efficiently exploited in several network implementations (as soon described).

Store-and-forward congestion has two unpleasant consequences: throughput degradation and deadlocks. These conditions were described in Sections II-A and II-C, respectively. In the remainder of this section, we survey and compare a number of hop level flow control procedures, specifically designed to eliminate these problems.

B. Classification of Hop Level Control Schemes

The hop level flow control scheme can play the role of arbitrator between various *classes of traffic* competing for a common buffer pool in each node. A fundamental distinction between different flow control schemes is based on the way the traffic entering a node is subdivided into classes.

One family of hop flow control schemes distinguishes incoming packets based on the output queue they must be placed into. Thus, the number of classes is equal to the number of the output queues; the flow control scheme supervises the allocation of store-and-forward buffers to the output queues. Some limit (fixed or dynamically adjustable) is defined for each queue; packets beyond this limit are discarded. Hence, the name *channel queue limit* schemes is generally given to such mechanisms (see Section III-C).

Another important family of hop flow control schemes distinguishes incoming packets based on the "hop count" (i.e., the number of network links that they have so far traversed). This implies that each node keeps track of $N - 1$ classes of traffic, where $N - 1$ = number of different hop counts, and N = the number of nodes in the network (note that if loopless routing is assumed, no network path can exceed $N - 1$ loops in length), and allocates a (fixed or adjustable) number of buffers to each class. We will refer to this family of schemes as *buffer class* schemes (see Section III-D).

A third family distinguishes packets based on the virtual circuit (i.e., end-to-end session) they belong to. This type of scheme requires, of course, a virtual circuit network architecture; it assumes that each node can distinguish incoming packets based on the virtual circuit they belong to and keep track of a number of classes equal to the number of virtual circuits that currently traverse it. Note that the number of classes varies here with time (since virtual circuits are dynamically created and realeased), as distinct from the previously mentioned schemes where the number of classes is merely a function of the topology. Upon creation, a virtual circuit is allocated a set of buffers (fixed or variable) at each node. When this set is used up, no further traffic is accepted from that virtual circuit. We will refer to this family of schemes as *virtual circuit hop level* schemes (see Section III-E).

Many other traffic subdivisions are possible: for example, a traffic class may be associated with each traffic source; with each traffic destination; or with each source-destination node pair. Indeed, these are all legitimate and, in many respects, well justified choices for a link level flow control scheme. However, we will restrict our study to the three schemes just mentioned, since these are the only schemes which have been extensively analyzed in the published literature and implemented in real networks.

Apart from traffic class distinctions, another parameter that is often used to characterize and classify hop flow control schemes is the degree of dynamic sharing of the store-and-forward buffers. Here, several possibilities exist, namely:

1) fixed, uniform partitioning of buffers among buffer classes (no sharing);

2) buffer partitioning proportional to traffic in each class (no sharing);

3) overselling (i.e., the sum of the buffer limits, one for each class, is larger than the total buffer pool); and

4) dynamic adjustment of buffer limits based on relative traffic fluctuations.

The following sections discuss each hop flow control class in more detail.

C. Channel Queue Limit Flow Control

In the channel queue limit (CQL) scheme, the traffic classes correspond to the channel output queues, and there are restrictions on the number of buffers each class can seize. We may define the following versions of the CQL scheme [20].

1) Complete Partitioning (CP)—Letting N = number of output queues, and n_i = number of packets on the ith queue and B = buffer size, we have the following constraint:

$$0 \leqslant n_i \leqslant \frac{B}{N}, \qquad \forall i.$$

2) Sharing with Maximum Queues (SMXQ)—Let $b_{\max}$ be the maximum queue size allowed (where, $b_{\max} > B/N$); we have the following constraints:

$$0 \leqslant n_i \leqslant b_{\max}, \qquad \forall i$$

$$\sum_i n_i \leqslant B.$$

3) Sharing with Minimum Allocation (SMA)—Let $b_{\min}$ be the minimum buffer allocation which is guaranteed to each queue (typically, $b_{\min} \leqslant B/N$). The constraint then becomes

$$\sum_i \max\,(0, n_i - b_{\min}) \leqslant B - Nb_{\min}.$$

4) Sharing with Minimum Allocation and Maximum Queue—This scheme combines 2) and 3) in that it provides for a minimum buffer guarantee and a maximum buffer allocation for each queue at the same time.

The above options assume that the buffer limit parameters are fixed in time and are the same for all queues. Additional flexibility may be introduced in these schemes by allowing the buffer parameters to change dynamically in time and from queue to queue based on traffic fluctuations.

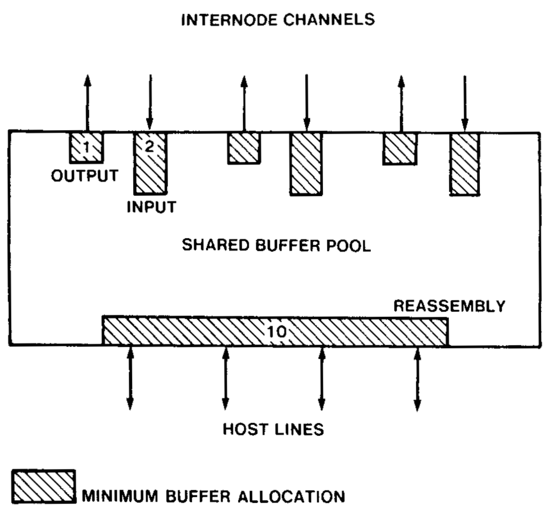
Having defined a number of CQL flow control options, we now proceed to show that this form of flow control can eliminate the performance degradation and deadlock effects mentioned in Section II. Referring first to Fig. 2, we note that in the presence of CQL flow control, the traffic component (B, B') will no longer be permitted to seize all the buffers in the switch. Therefore, traffic can now flow freely from A to A', and the throughput degradation effect is removed. Similarly, the deadlock condition depicted in Figs. 6 and 7 cannot occur since the buffers in node A cannot be taken over completely by the channel (A, B) queue. Therefore, some buffers in A will always be available to receive packets from node B.

Some form or another of CQL flow control is found in every network implementation. The ARPANET IMP (interface message processor) has a shared buffer pool with minimum allocation and maximum limit for each queue, as shown in Fig. 12 [30]. Of the total buffer pool (typically, 40 buffers), two buffers for input and one buffer for output are permanently allocated to each internode channel. Similarly, ten buffers are permanently dedicated to the reassembly of messages directed to the hosts. The remaining buffers are shared among output queues and the reassembly function, with the following restrictions: reassembly buffers $\leqslant 20$, output queue $\leqslant 8$, the total store-and-forward buffers $\leqslant 20$.

Next we proceed to the evaluation and comparison of CQL implementations, and briefly review the main results available in the published literature [18], [20]. We first report on some throughput degradation conditions observed in absence of flow control. Fig. 13 from [18] shows throughput performance as a function of link load for a variety of buffer control policies. The curve labeled "unrestricted sharing" corresponds to a system without flow control. We notice that, for increasing input load, the throughput of the uncontrolled system reaches a peak and then degrades asymptotically to unity. This behavior confirms the throughput degradation predictions made in Section II.

Throughput degradation is easily corrected with the introduction of CQL flow control, as shown by the remaining curves in Fig. 13. The "no sharing" system (i.e., complete partitioning of the buffer pool among the outgoing queues) is, as expected, the most conservative and throughputwise least efficient scheme. The best scheme is the "optimal sharing" scheme, which corresponds to optimally reselecting a new buffer limit for each level of traffic (i.e., dynamic SMXQ). A heuristic approximation of the optimal scheme is offered by the "square root scheme," a load invariant scheme with fixed buffer limit $=B/N$, where B = total number of buffers and N = number of output channels. The square root scheme is simpler to implement than the optimal scheme since it does not depend on traffic load and, therefore, does not require the reoptimization of the buffer limit values as a function of traffic pattern changes, and yet, it was shown to be practically as efficient as the optimal sharing for a number of cases [18].

Kamoun [20] used a similar switch model to investigate the sharing with minimum allocation (SMA) scheme. The results, obtained in a balanced load environment, show no substantial difference between SMXQ and SMA; in fact, neither scheme is consistently better over the entire range of offered loads. We conjecture, however, that with strongly unbalanced traffic SMA would exhibit better "fairness" since SMA guarantees minimum throughput (with low delay) for each output channel even when the shared portion of the buffer pool is captured by a few heavily loaded queues.



Total buffer pool = 40 buffers

	minimum allocation	maximum allocation
Reassembly	10	20
Internode input queue	2	
Internode output queue	1	8
Total internode queues (i.e., total S/F buffers)		20

Fig. 12. Buffer allocation in Arpanet IMP (1972 version).

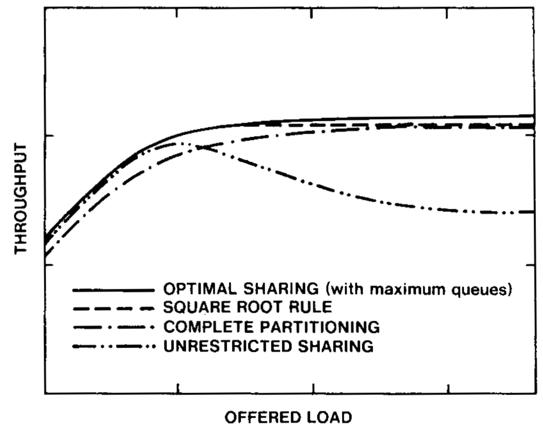


Fig. 13. Single switch buffer and allocation model. Throughput versus load behavior for various buffer management schemes (unbalanced load pattern).

Summarizing various published results, we may state that CQL flow control is necessary to avoid throughput degradation, unfairness, and direct store-and-forward deadlocks. Furthermore, it seems that almost any form of CQL implementation will provide the minimum required protection. The safest scheme (for fairness reasons) seems to be the combination of SMXQ and SMA, which imposes a maximum and minimum limit on each queue (incidentally, this was the scheme used in ARPANET).

D. Structured Buffer Pool (SBP) Flow Control

We have shown in the previous section that CQL flow control eliminates direct store-and-forward deadlocks. However, there is another, more general form of deadlock which can arise in packet networks, namely, *indirect store-and-forward deadlocks* [19]. Fig. 14 illustrates a typical indirect store-and-forward deadlock situation. Suppose that unfavorable traffic conditions in the ring topology shown in Fig. 14 cause each queue to be filled with $Q_{\max}$ packets, where $Q_{\max}$ is the limit imposed by the CQL strategy. Furthermore, assume that the packets at each node are directed to a node two or more hops away (e.g., all packets queued on link (A, B) are directed to C). In these conditions, no traffic can move in the network since all the queues are full. Thus, we have a deadlock even if the network is equipped with CQL flow control (which is known to prevent direct store-and-forward deadlocks)!

Prevention of indirect store-and-forward deadlocks is obtained with the "structured buffer pool" strategy proposed by Raubold *et al.* [37]. In this strategy, packets arriving at each node are divided into classes according to the number of hops they have covered. For example, packets entering a node from the host belong to class 0 of that node, since they have not yet covered any hops. The highest class $H_{\max}$ corresponds to packets that have traversed $H_{\max}$ hops, where $H_{\max}$ is the maximum path length in the network (a function of the topology and the routing algorithm). The highest class $H_{\max}$ also includes all the packets that have reached their destinations and are therefore being reassembled into messages before delivery to the hosts. The nodal buffer organization reflects this class structure as shown in Fig. 15. Each packet class has the right to use a well-defined set of buffers. Class 0 can access only the buffers available in set 0. Buffer set 0 is large enough to store the largest size message entering the network. Class $i + 1$ can use all the buffers available to class i, plus one additional buffer. Finally, class $H_{\max}$ can access all the buffers available to class $H_{\max}-1$, plus a number of buffers sufficient to reassemble the largest message to be delivered to any destination (this provision is necessary, although not sufficient, to avoid "reassembly deadlocks," as will be shown in Section IV).

Under normal traffic conditions, only set 0 buffers are used. When the load increases beyond nominal levels, buffers fill up progressively from level 0 to level $H_{\max}$. When at a given node the buffers at levels $\leqslant i$ are full, arriving packets which have covered $\leqslant i$ hops are discarded. Thus, in case of congestion, "junior" packets are dropped in the attempt to carry "senior" packets to their destination. This is a desirable property, since senior packets correspond to a higher network resource investment.

It can easily be shown that this strategy eliminates deadlocks of both the direct and indirect type [37]. To prove this, we consider the "resource graph" [3] associated with the packet switch network. In this graph, there is an arc associated with each packet in the network. The arc originates from the buffer currently occupied by the packet and terminates in the (currently unavailable, but awaited) buffer in the next node on the path. A deadlock occurs if and only if there is a cycle in the graph, i.e., there is a chain of arcs which starts from one buffer, and terminates at the same buffer. The existence of cycles can easily be recognized in the deadlock situations depicted in Figs. 6 and 14.

With the structured buffer pool, however, no cycle can occur in the resource graph since each arc starts from a buffer of class i and points to a buffer of class $i + 1$ (recall that a

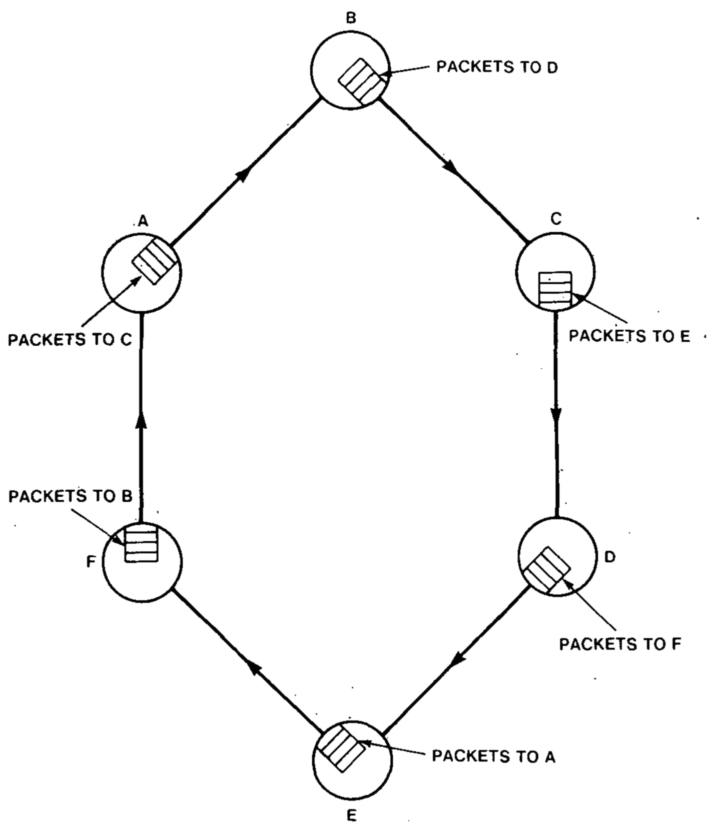
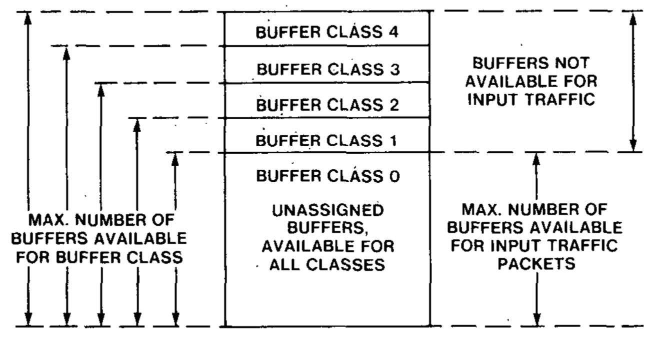


Fig. 14. Indirect store-and-forward deadlock.


BUFFER CLASS 4
BUFFER CLASS 3
BUFFER CLASS 2
BUFFER CLASS 1
BUFFER CLASS 0
UNASSIGNED BUFFERS, AVAILABLE FOR ALL CLASSES
BUFFERS NOT AVAILABLE FOR INPUT TRAFFIC
MAX. NUMBER OF BUFFERS AVAILABLE FOR INPUT TRAFFIC PACKETS
MAX. NUMBER OF BUFFERS AVAILABLE FOR BUFFER CLASS
Fig. 15. Structured buffer pool.

packet gains seniority at each hop; an illustration of this property is shown in Fig. 16. Thus, both direct and indirect store-and-forward deadlocks are prevented.

The SBP method was developed by the GMD group in Darmstadt, Germany, for implementation in GMDNET, an experimental packet switch network [37]. Before implementation, an extensive simulation effort was carried out to verify and evaluate the performance of all the network protocols, and of the SBP procedure in particular [13]. Early simulation results showed that the proposed flow control scheme was effective in eliminating deadlocks, but was not successful in preventing throughput degradation when the offered load exceeded the critical threshold (some SBP simulation experiments with typical packet switch network topologies showed that the throughput in heavy load conditions was four to five times lower than the maximum throughput).

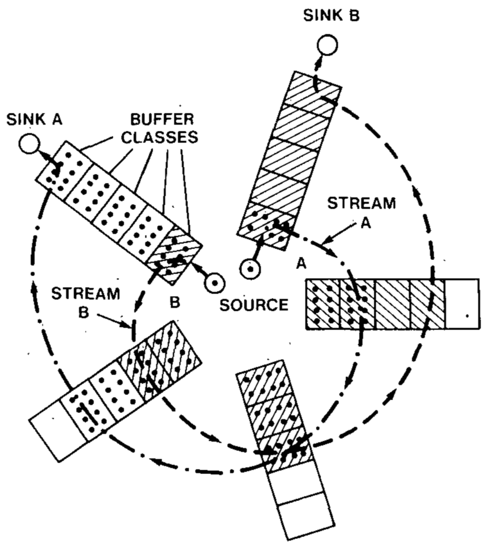


Fig. 16. Access to buffer classes. Example for two data streams. Dotted area: buffers available for stream A; hatched area: buffers available for stream B.

To correct the loss of throughput efficiency under heavy loads, additional constraints were imposed on the number of buffers that each traffic class could seize. The most dramatic improvement was obtained by limiting the number of class 0 buffers that could be seized by input packets (i.e., packets entering the network from external sources). In the absence of this constraint, input packets had the tendency to monopolize all class 0 buffers, leaving only a "thin" buffer layer for the transit traffic to circulate. The control of input traffic, known as "input flow control" in GMDNET is a form of network access flow control and will be discussed more extensively in Section V.

Additional improvements in the SBP scheme were obtained in the case of datagram networks, by setting a specific buffer size constraint $L(i)$ on each class i [13]. (In other words, instead of having a nested buffer pool in which class i can access all buffers available to class $i - 1$, plus one buffer, a different constraint is set on each class). The constraint $L(i)$ was dynamically adjusted to adapt to the relative demands of the various classes. It is interesting to note that the deadlock prevention property is *not* affected by dynamic changes in buffer class size (as long as at least one buffer is dedicated to each class at all times).

E. Virtual Circuit (Hop Level) Flow Control

We recall that packet switch networks can be subdivided into two broad classes: datagram (DG) networks and virtual circuit (VC) networks. In DG networks, each packet in a user session is carried through the network independently of the other packets in the same session; that is, packets in the same session may follow different routes, and may be delivered out of sequence to the destination. In VC networks, a physical network path is set up for each user session and is released when the session is terminated. Packets follow the preestablished path in sequence. Sequencing and error control are provided at each step along the path.

The previously mentioned flow control schemes, namely CQL and SBP, are applicable to both DG and VC nets. In addition, VC nets permit the application of *selective flow control* to each individual VC stream (VC flow control). There are two forms of VC flow control:

1) hop level (or stepwise) VC flow control, which controls VC flow at each hop along the path, and is designed to avoid S/F buffer congestion; and

2) source-sink (or end-to-end) VC flow control, whose function it is to adjust source rate to sink rate so as to maximize VC throughput, yet avoiding sink buffer congestion.

In this section we will mainly deal with VC hop level (VC-HL) flow control; we discuss end-to-end VC flow control in more detail in Section IV.

The basic principle of operation of the VC-HL scheme consists of setting a limit M on the maximum number of packets for each VC stream that can be in transit at each intermediate node. The limit M may be fixed at VC setup time, or may be dynamically adjusted, based on load fluctuations. The buffer limit M is enforced at each hop by the VC-HL protocol, which regulates the issue of transmission "permits" and discards packets based on buffer occupancy.

The advantage of VC-HL (over CQL and SBP) is to provide a more efficient and prompt recovery from congestion by selectively slowing down the VC's directly feeding into the congested area. By virtue of backpressure, the control then propagates to all the sources that are contributing to the congestion, and reduces (or stops) their inputs, leaving the other traffic sources undisturbed. Without VC-HL flow control, the congestion would spread gradually to a larger portion of the network, blocking traffic sources that were not directly responsible for the original congestion, and causing unnecessary throughput degradation and unfairness.

As in the case of CQL and SBP schemes, various buffer sharing policies can be proposed. At one extreme, M buffers can be dedicated to each VC at setup time; at the other extreme, buffers may be allocated, on demand, from a common pool (complete sharing). It is easily seen that buffer dedication can lead to extraordinary storage overhead, since there is, generally, no practical upper bound on the number of VC's that can simultaneously exist in a network; furthermore, the traffic on each VC is generally bursty, leading to low utilization of the reserved buffers. For these reasons, most of the implementations employ dynamic buffer sharing.

The shared versus dedicated buffer policy also has an impact on the deadlock prevention properties of the VC-HL scheme. With buffer dedication, the VC-HL scheme becomes deadlock free. This can easily be deduced by considering the resource graph and recognizing that the graph cannot contain loops, since virtual circuits are loopless by construction. (For deadlock freedom, it actually suffices that at least one buffer be reserved for each virtual circuit). If, on the other hand, no buffer reservations are made and buffers are allocated strictly on demand, deadlocks may occur unless additional protection (e.g., the SBP scheme) is implemented.

In the following, we briefly describe three different versions of VC-HL flow control implemented in existing networks, and report on some performance results.

Tymnet is probably the earliest VC network developed [39]. As distinct from most VC networks, Tymnet uses a "composite" packet internode protocol. This means that data from different VC's traveling on the same trunk can be packed in the same envelope, for the purpose of link overhead reduction. Tymnet is a character-oriented network in the sense that data flows on the virtual circuit in the form of characters, rather than packets (i.e., characters are assembled into packets at the entry node, and are then disassembled at the exit node). The character-oriented nature of Tymnet implies that VC-HL buffer allocation is based on character (rather than packet) counts.

In Tymnet [39], a throughput limit is computed for each VC at setup time according to terminal speed, and is enforced all along the network path. Throughput control is obtained by assigning a maximum buffer limit (per VC) at each intermediate node and by controlling the issue of transmission permits from node to node based on the current buffer allocation. Periodically (every half second), each node sends a backpressure vector to its neighbors, containing one bit for each virtual circuit that traverses it. If the number of currently buffered characters for a given VC exceeds the maximum allocation (e.g., for low speed terminals—10 to 30 cps—the allocation is 32 characters), the backpressure bit is set to zero; otherwise the bit is set to one. On the transmitting side, each VC is associated with a counter which is initialized to the maximum buffer limit and is decremented by one for each character transmitted. Transmission stops on a particular VC when the corresponding counter is reduced to zero. Upon reception of a backpressure bit = 1, the counter is reset to its initial value and transmission can resume.

The effect of backpressure from an individual hop back along the VC in Tymnet constitutes a good example of the "hybrid" character of many practical flow control implementations, since we see here a mixture of hop level and transport level flow control. This was pointed out earlier in connection with Fig. 10, and we shall encounter other examples as we proceed.

Transpac, the French public data network, is a VC network which uses X.25 as an internode protocol [42]. One of the distinguishing features of Transpac is the use of the throughput class concept in X.25 for internal flow and congestion control. Each VC call request carries a throughput class declaration which corresponds to the maximum (instantaneous) data rate that the user will ever attempt to present to that VC. Each node keeps track of the aggregate declared throughput (which represents the worst case situation), and at the same time, monitors actual throughput (typically, much lower than the declared throughput) and average buffer utilization. Based on the ratio of actual to declared throughput, the node may decide to *oversell* capacity, i.e., it will attempt to carry a declared throughput volume higher than trunk capacity. Clearly, overselling implies that input rates may temporarily exceed trunk capacities, so that the network must be prepared to exercise flow control. Packet buffers are dynamically allocated to VC's based on demand (complete sharing), but thresholds are set on individual VC allocations as well as on overall buffer pool utilization.

Of particular interest is the impact of overall buffer pool thresholds on VC-HL. Three threshold levels [S_0, S_1, and S_2 (where $S_0 < S_1 < S_2$)] are defined and are used in the following way:

1) S_0: do not accept new VC call requests;

2) S_1: slow down the flow on current VC's (by delaying the return of ACK's at the VC level); and

3) S_2: selectively disconnect existing VC's.

The threshold levels S_0, S_1, and S_2 are dynamically evaluated as a function of declared throughput, measured throughput and current buffer utilization.

Another example of a VC network is offered by GMDNET [13]. As we mentioned before, GMDNET applies SBP flow control. In addition, it applies I-control (individual control) on each virtual circuit. I-control consists of two components: end-to-end flow control and hop level flow control. End-to-end and hop level flow control are implemented using variable size windows PUL_E and PUL_L, respectively (PUL = packet underway limit). The window is defined as the maximum number of packets that a sender is allowed to transmit before receiving an ACK, or permit [5]. The windows PUL_E and PUL_L are dynamically adjusted based on sink congestion and intermediate node congestion, respectively; their values may vary within predefined ranges ($1 \leqslant \text{PUL}_E \leqslant W_E$; $1 \leqslant \text{PUL}_L \leqslant W_L$) [37], [13]. The buffer pool is completely shareable, without specific reservations for individual VC's.

Simulation results on the performance of the I-control scheme lead to the following important conclusions.

1) I-control alone cannot prevent throughput degradation, unfairness, and deadlocks. Experimental results clearly show that an I-controlled network without SBP becomes deadlocked immediately after the applied load exceeds the critical value (this confirms our prediction that VC flow control without specific buffer reservations for individual VC's cannot prevent deadlocks).

2) The end-to-end component of I-control is very effective in preventing network congestion in the case of source rates exceeding sink rates. Without I-control (i.e., the SBP control alone), a five-fold throughput degradation was observed in a typical network experiment.

IV. ENTRY-TO-EXIT FLOW CONTROL

The main objective of the entry-to-exit (ETE) flow control is to prevent buffer congestion at the exit node due to the fact that remote sources are sending traffic at a higher rate than can be accepted by the hosts (or terminals) fed by the exit node. The cause of the bottleneck could be either the overload of the local lines connecting the exit node to the hosts, or the slow acceptance rate of the hosts. The problem of congestion prevention at the exit node becomes more complex when this node must also reassemble packets into messages, and/or resequence messages before delivery to the host. In fact, reassembly and resequence deadlocks may occur, which require special prevention measures.

In order to understand how reassembly deadlocks can be generated, let us consider the network path shown in Fig. 17, where three store-and-forward nodes (node 1, node 2, and node 3, respectively) relay traffic directed to host 1. In the situation depicted in Fig. 17, three multipacket messages A, B, and C are in transit towards host 1. Without loss of generality we assume that the message size is $\leqslant 4$ packets and that 4 buffers are dedicated to messages being assembled at a node; furthermore, a channel queue limit $Q_{\max} = 4$ is set on each trunk queue, for hop level flow control. We note from Figure 17 that message A (which has siezed all four reassembly buffers at node 3) cannot be delivered to the host since packet A_2 is missing. Packet A_2, on the other hand, cannot be forwarded to node 2 since the queue at node 2 is full. The node 2 queue, in turn, cannot advance until reassembly space becomes available in node 3 for B or C messages. Deadlock!

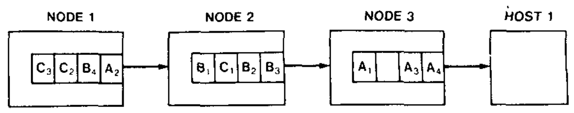


Fig. 17. Reassembly buffer deadlock.

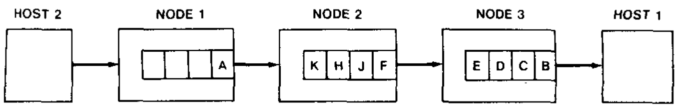


Fig. 18. Resequence deadlock.

A very similar order of events leads to resequence deadlocks as shown in Fig. 18. Assume that a sequence of single packet messages A, B, $\cdots$, K originating from host 2 and directed to host 1 is traveling through a three-node network. If messages must be delivered in sequence, messages B, C, D, E in node 3 cannot be transmitted to host 1 until message A is received at node 3. However, due to store-and-forward buffer unavailability in node 2, message A cannot reach node 3. Deadlock!

Various schemes can be used to prevent these types of deadlocks. In the ARPANET, for example, reassembly deadlocks are avoided by requiring a reassembly buffer reservation for each multipacket message entering the network; resequence deadlocks are avoided by discarding out-of-sequence messages at the destination. Other networks (e.g., Telenet) have sufficient nodal storage to permit out-of-sequence messages to be accepted at a destination node with the understanding that these may be discarded later if storage congestion occurs; again, the existence of a source copy saves the day. These and other schemes are discussed in more detail in the following sections.

While the main objective of ETE controls is to protect the exit node from congestion, an important byproduct is the prevention of global (i.e., internal) congestion. Virtually all ETE controls are based on a window scheme that allows only up to W sequential messages to be outstanding in the network before an end-to-end ACK is received. If the network becomes congested (this may occur independently of destination node congestion), messages and ACK's incur high end-to-end delays. These delays, combined with the restriction on the total number of outstanding messages, effectively contribute to reduce the input rate of new packets into the network.

Several varieties of ETE flow control schemes have been proposed and implemented. We first describe four representative examples, and then briefly review some analytical and simulation models for the performance evaluation and comparison of such schemes.

A. ARPANET RFNM and Reassembly Scheme

ETE flow control in ARPANET is exercised on a host-pair basis [30], [23]. Specifically, all messages traveling from the same source host to the same destination host are carried on the same logical "pipe." Each pipe is individually flow controlled by a window mechanism. An independent message number sequence is maintained for each pipe. Numbers are sequentially assigned to messages flowing on the pipe, and are checked at the destination for sequencing and duplicate detection purposes. Both the source and the destination keep a small window w (presently, $w = 8$) of currently valid message numbers. Messages arriving at the destination with out-of-range numbers are discarded. Messages arriving out of order are discarded since storing them (while waiting for the missing message) may lead to potential resequence deadlocks. Correctly received messages are acknowledged with short ETE control messages called RFNM's (ready for next message). Upon receipt of an RFNM, the sending end of the pipe advances its transmission window, accordingly.

RFNM's are also used for error control. If an RFNM is not received after a specified time out (presently about 30 s), the source IMP sends a control message to the destination inquiring about the possiblity of an incomplete transmission. This technique is necessary to keep source and destination message numbers synchronized and also to request a retransmission from the host in the case of message loss.

The window and message numbering mechanisms described so far support ETE flow control, sequencing and error control functions in the ARPANET. A separate mechanism, known as reassembly buffer allocation [30], is used to prevent reassembly deadlocks. Each multipacket message must secure a reassembly buffer allocation at the destination node before transmission. This is accomplished by sending a reservation message called a REQALL (request for allocation) to the destination and waiting for an ALL (allocation) message from the destination before attempting transmission. To reduce delay (and, therefore, increase throughput) of steady multipacket message flow between the same source-destination pair, ALL messages are automatically piggybacked on RFNM's, thus eliminating the reservation delay for all messages after the first one. If a pending allocation at the source node is not claimed within a given time-out (250 ms), it is returned to the destination with a "giveback" message. Single packet messages are transmitted to their destinations without buffer reservation. However, if upon arrival at the destination, all the reassembly buffers are full, the single packet message is discarded and a copy is retransmitted from the source IMP after an explicit buffer reservation has been obtained. Some pitfalls inherent in such schemes are described in [23].

B. SNA Virtual Route Pacing Scheme

The IBM systems network architecture (SNA) is an architecture aimed at providing distributed communications and distributed processing capabilities between IBM systems [15], [16]. SNA was first announced in 1974. Since then, the original set of functions which supported single rooted networks (i.e., single host) have been enhanced to support multiple-domain (i.e., multiple host) networking. In this paper, we refer to SNA release 4.2 [16].

SNA devices can be subdivided into four main categories: host computers (e.g., system/370), communications controllers (e.g., 3704 and 3705), terminal cluster controllers, and terminal devices (e.g., TTY's, CRT's, readers, and printers). Distributed communications with full routing, flow control, and global addressing capabilities are provided only on store-and-forward networks interconnecting host computers and communication controllers. These nodes are called subarea nodes in SNA. Terminals and terminal cluster controllers are connected into this high level at these subarea nodes, which provide the necessary boundary functions (e.g., global/local address conversion, etc.). Thus, for purposes of this section, SNA can be viewed as the usual two-level network architecture, with terminals and terminal cluster controllers at the lower level, and hosts and communications controllers at the higher level.

SNA is essentially a virtual circuit network, in the sense that each user session is associated with a physical route at session setup time. The routing policy is a static, multipath policy which maintains up to eight distinct routes between each source-destination pair in the high-level network (i.e., between subarea nodes). These routes are called ER's (explicit routes), to distinguish them from VR's (virtual routes) defined below. ER's are defined as an ordered sequence of network trunks, and are uniquely identified by ER numbers. When a failure is detected on an ER currently being used, the next ER on the list is then "switched in." One difficulty here is that the list of ER's must be established by the network designer each time the network topology is changed.

Next, virtual routes (VR's) are defined between each source-destination node pair of the high level network. A VR is essentially a virtual pipe which is constructed on top of an ER and is subject to flow control. Three sets of VR's, each with a different level of priority are maintained between each subarea node pair. Each set may consist of up to eight VR's, thus allowing for up to 24 VR's between each high level network node pair. Active VR's are identified by VR numbers and are stored in lists at each node.

At setup time, the entry node scans the VR list and assigns the user session to the first available virtual route of desired priority. Several user sessions may be multiplexed on the same VR. In turn, several VR's may be multiplexed on the same ER. Finally, several ER's can be multiplexed on the same trunk.

The rationale for the distinction between virtual routes and explicit routes (a unique SNA feature among all VC networks, which typically associate a virtual route with a fixed path) is to "...insulate the virtual route layer from the physical configuration" [16]. As a consequence, user packets are driven through the network using the ER ID number, while the VR ID number needs to be checked only at the endpoints of the path. This feature considerably reduces storage and processing overhead with respect to conventional VC schemes, which typically require large maps at each intermediate node to store the information relative to all virtual circuits traversing that node.

In the high-level network, flow control is applied independently to each VR from entry to exit node. This scheme, known as *VR pacing* is actually a combination of ETE and hop level flow control. It is based on a window mechanism, in which the entry node must request (and obtain) permission from the exit node before sending a new group of k packets, where k = window size. The destination may grant (or delay) such permission depending on local buffer availability. The window size k varies from h to $3h$, where h is the path hop length. The value of k is dynamically adjusted not only by the exit node, but also by any intermediate node along the path on the basis of its buffer availability [1]. The fact that both the end node and the intermediate nodes can "modulate" the window size k makes VR pacing a hybrid ETE and hop flow control scheme.

In addition to VR-pacing control, which operates between subarea nodes, the SNA architecture provides also for *session level pacing* which, for terminals, extends beyond subarea nodes and individually flow controls each user session between terminal and host computer. Session pacing is discussed in Section VI.

C. GMD Individual Flow Control

In GMDNET, entry-to-exit flow control is exercised individually on each virtual circuit, hence the name of individual flow control assigned to the scheme [37]. We recall that GMDNET is a VC network in which a fixed route is assigned to each user session at session setup time.

The main purpose of entry-to-exit flow control in GMDNET is to protect the exit node from overflow caused by low sink rates. When the source host rate exceeds the sink host rate, the flow control mechanism intervenes to slow down inputs from the source host into the entry node. This is achieved by maintaining a window of outstanding packets between entry and exit node for each virtual circuit. The window must be large enough to permit each virtual circuit to efficiently utilize the bandwidth available on the path. GMD simulation experiments have shown that $w = h + 1$ (where h = hop length of the path) is a satisfactory choice under nominal load conditions. Window size can be reduced if the sink is slow in accepting packets. More precisely, when for a given VC the queue waiting to be transferred from exit node to sink reaches the value w, further arrivals to the exit node within that VC are discarded and a negative ACK is returned to the source node. Each negative ACK causes a window size reduction of 1 at the source node, until the minimum window size $w = 1$ is reached. Each positive ACK, on the other hand, increases window size by 1, until the maximum window size $w = h + 1$ is reached. In this way, window size is dynamically controlled in the range 1 to $h + 1$ by positive and negative acknowledgments [37].

In addition to the entry-to-exit flow control, each hop of the virtual circuit is also independently flow controlled (see Section III). The two layers of flow control, entry-to-exit and hop, are logically separated one from the other, in that the ETE window is controlled by exit buffer occupancy, while hop window is controlled by intermediate node congestion.

Packets within the same virtual circuit must be delivered to the host in sequence, and in case of multipacket messages, must be reassembled before delivery to the host. Fixed path routing and link level sequencing imply that packets arrive at their destination in sequence. This sequencing property, and the fact that a number of buffers sufficient to reassemble the largest size packet is permanently dedicated to traffic leaving the network, preclude the possibility of reassembly deadlocks and eliminate the need for reassembly buffer allocation schemes of the type implemented in ARPANET.

D. Datapac Virtual Circuit Flow Control

The Canadian public data network, Datapac, implemented with the Northern Telecom SL-10 Packet Switching System provides virtual circuit services using an internal transport protocol built on top of a datagram subnetwork [28]. Flow control is exercised from entry to exit node on a virtual circuit basis, although no physical path is actually assigned to each virtual circuit, as was the case with SNA and GMDNET. The absence of a fixed path leads to some complications in the resequencing and loss recovery procedures, which will soon be discussed.

In Datapac, a virtual circuit is provided between the two endpoints of each user session. The virtual circuit is implemented as the concatenation of three protocol segments: a packet level X.25 protocol from the source device (i.e., data terminating equipment or DTE) to entry node (i.e., data communications equipment or DCE), an internal protocol from entry DCE to exit DCE, and a packet level X.25 protocol from exit node (DCE) to destination node (DTE). Each one of these protocol segments is flow controlled by a window mechanism. Of particular interest to us is the fact that window controls on these three segments are synchronized so as to provide a means of matching source DTE transmission rate with destination DTE acceptance rate. Window control synchronization is achieved by withholding the return of ACK's on a window if the downstream window is full.

As an example, let us assume that all windows are of size $w = 3$, and that the window between entry and exit DCE is full (i.e., there are three outstanding packets). The next packet arriving from the source DTE to the entry DCE will be accepted (assuming buffer space is available), but will not be immediately acknowledged; rather, the ACK will be withheld until an ACK from the exit DCE is received, thus opening up the downstream window [28].

Within the concatenated window mechanism the entry-to-exit flow control serves the function of promptly reflecting back to the source an exit segment congestion situation by withholding ACK's. Recall that in GMDNET the entry-to-exit flow control provided a similar service by dynamically adjusting the window with positive or negative ACK's. In Datapac, things are complicated, however, by the fact that the window mechanism is used not only for flow control, but also for sequencing, packet loss recovery, and duplicate detection. These latter functions are not required in the GMDNET, since sequencing is enforced there by the fixed path routing policy, and packet loss could occur only if a node along the path failed, in which case the virtual circuit would be automatically reinitialized.

The use of window ACK's for loss recovery in Datapac leads to the following problem. If the exit DCE does not return to the entry DCE an ACK for a correctly received packet (because the exit segment is congested), the entry DCE will retransmit the packet after a time-out, under the assumption that the packet was lost (or was dropped by the exit DCE for lack of resequence space). If no ACK is received after a specified number of retransmissions, the entry DCE will clear the virtual circuit. In order to minimize the generation of duplicate packets, the value of time-out must be carefully selected as a function of window size and other network parameters.

E. Performance Models

The great majority of entry-to-exit flow control mechanisms are based on the window scheme. Critical parameters in the window implementation are the size of the window, and if error and loss recovery are to be provided, the retransmission time-out interval. Several analytic and simulation models have been developed recently to investigate the impact of these parameters on throughput and delay performance. This section briefly surveys some of the most significant contributions in this area.

We start with the Kleinrock and Kermani model of a single source-to-destination stream flow controlled by a window mechanism [26]. The network entry-to-exit delay is simplified as an $M/M/1$ queue delay, and the round trip delay therefore follows an Erlang-2 distribution. (This approximation is supported by simulation experiments showing that more accurate delay assumptions do not significantly change the nature of the results.) The exit node has finite storage and delivers packets to the destination host on a finite capacity channel. Consequently, the exit node may occasionally overflow and drop packets. To provide for transmission integrity, the entry node will retransmit an unacknowledged packet after a time-out interval. This simplified window model is solved analytically, yielding the optimal (i.e., minimum delay) window size and time-out interval for a given throughput requirement and destination buffer storage size.

In a subsequent paper [22], the same authors propose an adaptive policy (the "look-ahead" policy) for the dynamic adjustment of window size to time-varying traffic rate. In the proposed policy, the window size is dynamically controlled by the queue size at the exit node. Numerical results show that the delay versus throughput performance of the adaptively controlled scheme is somewhat superior to the performance of a scheme operated under static control, in which the window is adjusted in accordance with the traffic volume. These results are very encouraging, and are consistent with simulation experiments on dynamic window control carried out in multinode networks [1], [13].

The models in [26], [22] approximate the network as a single queue and therefore do not offer insight into the dependence of window size w on the number of intermediate hops. This issue is addressed by a simple multihop model developed by Kleinrock in [24]. In this model a packet stream from a single destination is transmitted across the network on a k-hop network path. Infinite buffer storage, and negligible error rates are assumed on each hop. The stream is flow controlled by a window mechanism. In this model, as the window size w increases, the end-to-end delay grows without limit while the throughput asymptotically reaches the path capacity. In order to find a meaningful criterion for the optimization of w, the concept of "power" as defined in Section II-F is used. We find that power is optimized by $w = k$. This implies that, at optimum, there should be on the average one packet in each intermediate queue. This result agrees with out intuition that the "entry-to-exit pipe should be kept full (in fact, *just* full)" for satisfactory performance. The general validity of this result is confirmed by actual window implementations. In fact, the SNA pacing scheme allows the window to dynamically vary from h to $3h$, where h = number of intermediate hops. Similarly, the GMD individual flow control scheme uses a maximum window of $h + 1$.

The main limitation of the two previous models is the single source, single destination traffic assumption which excludes interference at a given node by other traffic traversing it. The model by Pennotti and Schwartz [32] includes the effect of interference in an approximate fashion in that it represents a virtual link situation in which end-to-end link traffic flowing on a multihop path must compete at each hop with external traffic. This is essentially a "one hop" interference model in which some external traffic λ is injected into one node along the path and is transmitted to the next node on the path, where it then is removed from the network. The purpose of this study is to evaluate the possible path congestion caused by an increase in the virtual link rate λ_0, both with and without flow control. Congestion is defined as the relative average increase in time delay experienced by external users due to an increase in λ_0, taking $\lambda_0 = 0$ as a reference. Without flow control, congestion rapidly grows to infinity even for moderate values of λ_0. By introducing end-to-end window control which limits to w the number of packets outstanding on the virtual link at any one time, congestion can be bounded for any value of λ_0. The value of the upper bound varies with w, and decreases for decreasing w, as expected.

As an alternative to window flow control, hop flow control was also implemented in the Pennotti and Schwartz model by setting a limit on the number of link packets that would be stored at each intermediate node [32]. This scheme exhibited essentially the same performance as the window scheme. The above experiments show that flow control (either window or hop) can be used effectively to maintain fairness in a multiuser environment with conflicting requirements; that is, by adjusting the window parameter w, one can balance the relative user throughputs as desired.

The previously mentioned model offers some insight into multiuser flow control, but suffers from the limitation that only one virtual circuit can be flow controlled at a time, the remaining traffic components being kept constant. To remove this limitation, a number of multiple source, multiple destination models with selectively controlled user pairs have been developed. These models combine ETE flow control with network access flow control, and therefore may be regarded as hybrid models. Wong and Unsoy analyze a simple 5-node network to which individual entry-to-exit window control as well as isarithmic control are applied [41]. The isarithmic scheme is a network access flow control scheme which controls the total number of packets allowed in the entire network (see Section V for additional details). The major finding of this study is the fact that isarithmic control alone is not enough to guarantee efficient network operation. In fact, under some unfavorable traffic situations, one node pair may capture most of the permits, starving other pairs and leading to unfairness and to overall performance degradation. Similar results were found by Price in a series of simulation experiments [36]. The problem is corrected by introducing individual entry-to-exit flow controls in addition to isarithmic control.

The exact analysis of multinode networks with individually controlled node pairs becomes impractical for topologies with more than five to six nodes because of the rapidly increasing computational complexity of exact solution techniques [41]. To circumvent this problem, Reiser recently proposed an approximate solution technique based on a mean value analysis which is computationally affordable even for large networks, and which reaches a typical accuracy of 5 percent in throughput and 10 percent in delay [38]. With this technique it is now possible to analyze the interaction of various flow control schemes in a much more realistic environment (i.e., large networks; varied traffic patterns) than was possible with previous methods. Important design problems such as the optimization of window parameters for all source-destination pairs in order to maximize network throughput (within given fairness constraints), now become approachable.

In spite of the previously mentioned advances in computational solution techniques, some window flow control issues are still too complex to be attached analytically. For example, the dynamic control of window size in a multinode network is not amenable to a network-of-queues model even with the approximate solution methods. In these cases, simulation is still the leading performance evaluation tool [13], [1], [36].

V. NETWORK ACCESS FLOW CONTROL

A. Objective

The objective of network access (NA) flow control is to throttle external inputs based on measurements of internal network congestion. Congestion measures may be local (e.g., buffer occupancy in the entry mode), global (e.g., total number of buffers available in the entire network), or selective (e.g., congestion of the path(s) leading to a given destination). The congestion condition is determined at (or is reported to) the network access points and is used to regulate the access of external traffic into the network.

NA flow control differs from LL and ETE flow control in that it throttles external traffic to prevent *overall internal buffer congestion*, while LL flow control limits access to a specific store-and-forward node to prevent *local congestion and store-and-forward deadlocks*, and ETE flow control limits the flow between a specific source-destination pair to prevent *congestion and reassembly buffer deadlocks at the destination*. As we mentioned earlier, however, both LL and ETE schemes indirectly provide some form of NA flow control by reporting an internal network congestion condition back to the access point either via the backpressure mechanism (LL scheme), or via the credit slowdown caused by large internal delays (ETE scheme).

Three NA flow control implementations will be discussed: the isarithmic scheme, a global congestion prevention scheme based on the circulation of a fixed number of permits [8]; the input buffer limit scheme, a local congestion scheme which sets a limit on the number of input packets stored at each node [27], [13]; and the choke packet scheme, a selective congestion scheme based on the delivery of special control packets of that name from the congested node back to the traffic sources [29].

B. The Isarithmic Scheme

Since the primary cause of network congestion is the excessive number of packets stored in the network, an intuitively sound congestion prevention principle consists of setting a limit on the total number of packets that can circulate in the network at any one time. An implementation of this principle is offered by the Isarithmic scheme proposed for the National Physical Laboratories network [8], [35].

The isarithmic scheme is based on the concept of a "permit," i.e., a ticket that permits a packet to travel from the entry point to the desired destination. Under this concept, the network is initially provided with a number of permits, several held in store at each node. As traffic is offered by a host to the network, each packet must secure a permit before admission to the high level node is allowed. Each accepted packet causes a reduction of one in the store of permits available at the accepting node. The accepted data packet is able to traverse the network, under the control of node and link protocols, until its destination node is reached. When the packet is handed over to the destination subscriber, the permit which has accompanied it during its journey becomes free and an attempt is made to add it to the permit store of the node in which it now finds itself.

In order to achieve a viable system in which permits do not accumulate in certain parts of the network at the expense of other parts, it is necessary to place a limit on the number of permits that can be held in store by each node. If then, because of this limit, a newly freed permit cannot be accommodated at a node (overflow permit), it must be sent elsewhere. The normal method of carrying the permit in these circumstances is to "piggyback" it on other traffic, be this data or control. Only in the absence of other traffic need a special permit-carrying packet be generated.

A simulation program was developed by NPL to evaluate the performance of the isarithmic scheme in various network configurations and in the presence of different network protocols [35]. The main conclusion of these simulation studies was that the isarithmic scheme is a simple congestion prevention mechanism which performs well in uniform traffic pattern situations, but may lead to unnecessary throughput restrictions, and therefore, to poor performance in the case of nonuniform, time-varying traffic patterns. In particular, in the presence of high bandwidth data transfers, there is the possibility that permits are not returned to the traffic sources rapidly enough to fully utilize network capacity (the "permit starvation" problem). This would be the case when the destination node redistributes the overflow permits randomly in the network. If, on the other hand, the destination systematically returns all the permits to the source, the source-destination pair may end up capturing most of the network permits, thus causing unfairness. Tradeoffs between different permit distribution schemes are investigated with an analytical model in [41]. Finally, a delicate problem in isarithmic control is the bookkeeping of permits, to avoid unauthorized generation or disappearance of permits.

In spite of the above limitations, the isarithmic scheme proved to be very effective in *weakly controlled* networks (namely, networks without hop level flow control), eliminating congestion and deadlocks that had occurred without flow control. Some simulation experiments were also carried out on networks with hop level flow control (specifically CQL), and with a simple form of local access control (one buffer on each output queue was reserved for store-and-forward traffic). For this class of networks (called *strongly controlled* networks), it was found that the network performance did not show congestion tendencies even without isarithmic control in the case of a fixed routing discipline. When the fixed discipline was replaced with an adaptive routing discipline, it was found that the network would become easily congested since the simple form of network access control implemented would not prevent external traffic from flooding all the queues in the entry node. Again, the introduction of the isarithmic scheme was successful in eliminating the congestion problem for the adaptive routing case [36].

Critical parameters in the isarithmic scheme design are the total number of permits P in the network and the maximum number of permits L that can be accumulated at each node (permit queue). Experimental results show that optimal performance is achieved for $P = 3N$, where $N =$ total number of nodes, and $L = 3$. An excessive number of permits in the network would lead to congestion. An excessive value of L would lead to unfairness, accumulation of permits at a few nodes, and throughput starvation at the others.

C. Input Buffer Limit Scheme

The input buffer limit (IBL) scheme differentiates between input traffic (i.e., traffic from external sources) and transit traffic, and throttles the input traffic based on buffer occupancy at the entry node. IBL is a *local* network access method since it monitors local congestion at the entry node, rather than global congestion as does the isarithmic scheme. Entry node congestion, on the other hand, is often a good indicator of global congestion because the well known backpressure effect will have propagated internal congestion conditions back to the traffic sources.

The function of IBL controls is to block input traffic when certain buffer utilization thresholds are reached in the entry node. This flow control approach clearly favors transit traffic over input traffic. Intuitively, this is a desirable property since a number of network resources have already been invested in transit traffic. This intuitive argument is supported by a number of analytical and simulation experiments proving the effectiveness of the IBL scheme.

Many versions of IBL control can be proposed. Here, we describe and compare four different implementations that have been experimentally evaluated.

The term input buffer limit scheme refers to a scheme restricting the number of buffers made available to input traffic and was first introduced by the GMD research group [37], [13]. The scheme proposed for GMDNET is a by-product of the nested buffer class structure used to allocate buffers to different classes of traffic. We recall from Section III-D that the ith traffic class consists of all the packets that have already covered i hops. Input traffic is assigned to class zero (zero hops covered). Traffic class zero is entitled to use buffer class zero, which is a subset of the nodal buffer pool (in general, class i is entitled to use all buffer classes $\leqslant i$). Thus, input packets are discarded when class zero buffers are full. The size of buffer class zero (referred to as input buffer limit) was found to have a significant impact on throughput performance under heavy loads. Simulation experiments indicate that for a given topology and traffic pattern there is an optimal input buffer limit which maximizes throughput for heavy offered load. The use of lower or higher limits leads to a substantial drop in throughput [13].

A version of IBL control that is simpler than the GMD version was proposed by Lam [27] and analytically evaluated in an elegant model. Only two classes of traffic—input and transit—are considered in this proposal. Letting N_T be the total number of buffers in the node and N_I the input buffer limit (where $N_I \leqslant N_T$), the following constraints are imposed at each node:

1) number of input packets $\leqslant N_I$, and
2) number of transit packets $\leqslant N_T$

The analytical results confirm simulation results independently obtained by the GMD group. There is an optimal ratio N_I/N_T, which maximizes throughput for heavy offered load, as shown in Fig. 19. A good heuristic choice for N_I/N_T is the ratio between input message throughput and total message throughput at a node. As shown in the figure, throughput performance does not change significantly even for relatively large variations of the ratio N_I/N_T around the optimal value, thus implying that the IBL scheme is robust to external perturbations such as traffic fluctuations and topology changes. One shortcoming of this model is that all nodes in the net

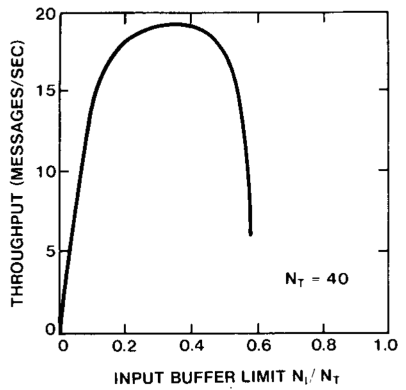


Fig. 19. Input buffer limit scheme: throughput versus buffer limitation for heavy offered load.

are assumed to have the same blocking probability, a somewhat unrealistic assumption.

A scheme similar to Lam's IBL scheme had been earlier proposed by Price [35]. In order to prevent input traffic from monopolizing the entire buffer pool, one buffer in each output queue was reserved for transit traffic. This is essentially equivalent to setting an input buffer limit $N_I = N_T - C$, where C = number of output channels. Simulation studies showed that this simple network access control based on source buffer utilization was quite successful in single level networks.

Kamoun [21] proposes yet another version of IBL control, in which an input packet is discarded if the *total* number of packets in the entry node exceeds a given threshold (whereas in Lam's scheme an input packet is discarded when the number of *input* packets exceeds a given threshold). Transit packets, instead, can freely claim all the buffers. The scheme is called drop-and-throttle flow control (DTFC) policy since a transit packet arriving at a full node is dropped and lost (loss model); while all previous schemes assumed link level retransmission of overflow packets (retransmit model). The DTFC scheme was analyzed using a network of queues model [21]. The results, shown in Fig. 20, clearly indicate that there is an optimal threshold value L which maximizes throughput for each value of offered load. Below the threshold, the network is "starved"; above the threshold, the network is congested. A similar scheme, referred to as the free flow scheme, is described and analyzed by Schwartz and Saad in [41]. Preliminary results indicate that, while free flow and IBL throughput performances are compatible, the free flow scheme offers substantial delay improvements.

We have pointed out that IBL control prevents congestion by favoring transit traffic over input traffic. In most cases (indeed, in all cases analyzed in the previously referenced studies), this favoritism leads to throughput improvements. In some cases, however, *unfairness* may result. Consider, for example, the 4-node network shown in Fig. 21. In this network, two file transfers, A to A' and B to B', respectively, are simultaneously competing for trunk (2, 3). Node 2 sees traffic A as transit traffic, so it gives it preferential treatment over traffic from B. Consequently, the A-A' packet stream can acquire more buffers in node 2, and thus achieve better throughput performance than the B-B' stream. The unfairness

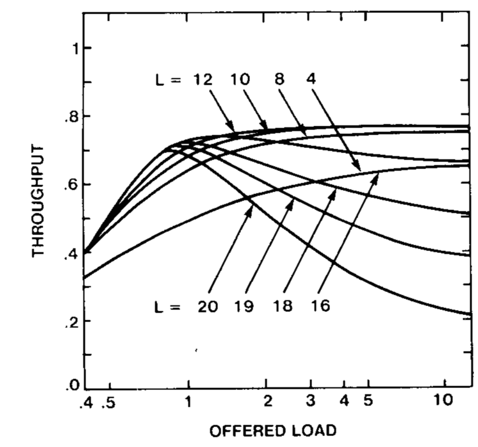


Fig. 20. Throughput versus load for a 121-node network for drop-and-throttle flow control.

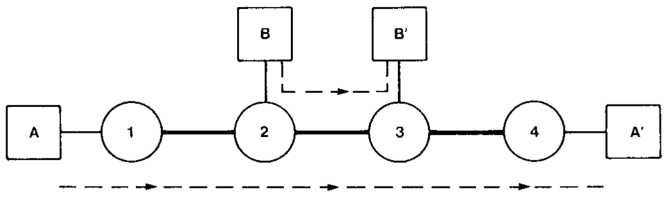


Fig. 21. Unfairness condition produced by input buffer limit and drop-and-throttle flow control schemes.

is particularly dramatic when DTFC is used. With the DTFC policy, if the A-packet queue in node 2 exceeds the buffer threshold (this could easily occur if $C_{23} < C_{12}$), B-packets cannot be accepted by node 2. Consequently B-traffic is completely shut off until the A-A' file transfer is completed.

D. Choke Packet Scheme

The choke packet (CP) scheme, proposed for the Cyclades network [29], is based on the notion of trunk and path congestion. A trunk (link) is defined to be congested if its utilization (measured over an appropriate history window with exponential averaging) exceeds a given threshold (e.g., 80 percent). A path is congested if any of its trunks are congested. Path congestion information is propagated in the network together with routing information and thus, each node knows hop distance and congestion status of the shortest path to each destination.

When a node receives a packet directed to a destination whose path is congested it takes the following actions.

1) If the packet is an *input* packet (i.e., it comes directly from a host), then the packet is dropped.

2) If the packet is a *transit* packet, it is forwarded on the path; but a "choke" packet (namely, a small control packet) is sent back to the source node informing it that the path to that destination is congested and instructing it to block any subsequent input packets to this destination. The path the to destination is gradually unblocked if no choke packets are received during a specified time interval.

This is a greatly simplified description of the CP scheme. Several other features (which are essential to make the scheme workable) are discussed in [29].

It is clear that the CP scheme attempts to favor transit traffic over input traffic, much in the same way as the IBL scheme did. The basic difference between the two schemes is the fact that IBL uses a local congestion measure, namely, the entry node buffer occupancy, to indiscriminately control all input traffic; whereas, CP uses a path congestion measure to exercise *selective* flow control on input traffic directed to different destinations.

Simulation experiments based on the Cigale network topology are given in Fig. 22 and show that the CP scheme can introduce substantial throughput improvements (with respect to the uncontrolled case) in sustained load conditions, asymptotically achieving the ideal performance for infinite load [29].

VI. TRANSPORT LEVEL FLOW CONTROL

A. Objectives

A transport protocol is a set of rules that govern the transfer of control and data between user processes across the network. The main functions of this protocol are the efficient and reliable transmission of messages within each user session (including packetization, reassembly, resequencing, recovery from loss, elimination of duplicates) and the efficient sharing of common network resources by several user sessions (obtained by multiplexing many user connections on the same physical path and by maintaining priorities between different sessions to reflect the relative urgency).

For efficient and reliable reassembly of messages at the destination host (or more generally, the DTE), the transport protocol must ensure that messages arriving at the destination DTE are provided adequate buffering. The transport protocol function which prevents destination buffer congestion and overflow is known as *transport level flow control.* Generally, this level of flow control is based on a "credit" (or window) mechanism as discussed earlier. Namely, the receiver grants transmission credits to the sender as soon as reassembly buffers become free. Upon receiving a credit, the sender is authorized to transmit a message of an agreed-upon length. When reassembly buffers become full, no credits are returned to the sender, thus temporarily stopping message transmissions [5].

The credit scheme described above is somewhat vulnerable to losses, since a lost credit may hang up a connection. In fact, a sender may wait indefinitely for a lost credit, while the receiver is waiting for a message. A more robust flow control scheme is obtained by numbering credits relative to the messages flowing in the opposite direction. In this case, each credit carries a message sequence number, say N, and a "window size" w. Upon receiving this credit, the sender is authorized to send all backlogged messages up to the $(N + w)$th message. With the numbered credit scheme, if a credit is lost then the subsequent credit will restore proper information to the sender [45].

Besides preventing destination buffer congestion, the credit scheme also indirectly provides global network congestion protection. In fact, store-and-forward buffer congestion at the intermediate nodes along the path may cause a large end-to-end credit delay, thus slowing down the return of credits to the sender, and consequently, reducing the rate of fresh messages input into the network.

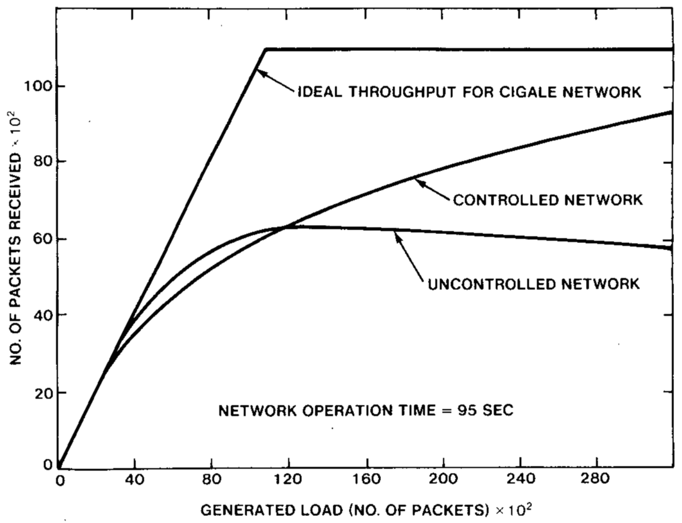


Fig. 22. Throughput performance in Cigale with and without flow control.

B. Implementations

Several versions of the transport protocol are in existence, each incorporating its own form of transport level flow control. Here, we briefly describe four representative implementations.

The earliest example of transport protocol implementation is the original version of the ARPANET network control program (NCP) [4]. NCP flow control is provided by unnumbered credits called "allocate" control messages (see Section IV-D). Only one allocate could be outstanding at a time (i.e., window size $W = 1$).

The French research network Cyclades provided the environment for the development of the transport station (TS) protocol [50]. In the TS protocol, the flow control mechanism is based on numbered credits, each credit authorizing the transmission of a variable size message called a letter. Flow control is actually combined with error control in that credits are carried by acknowledgment messages.

The transmission control program (TCP) was a second generation transport protocol developed by the ARPANET research community in order to overcome the deficiencies of the original NCP protocol [5]. As in the TS protocol, flow and error control are combined in TCP. As a difference, however, error and flow control are on a byte (rather than letter) basis. This allows a more efficient utilization of reassembly buffers at the destination.

In SNA, the transport level flow control is provided by session pacing. The purpose of session-level pacing is to prevent one session end from sending data more quickly than the receiving session end can process the data [16]. As in TCP and TS, session-level pacing is based on a window concept, in which the receiving end grants "credits" to the sending end based on its buffer availability and processing capability. As a difference, however, subarea nodes in SNA can control the inbound flow from a cluster controller into the network by withholding the credits (called pacing responses in SNA) for a given session, if the subarea node buffers are congested or if the Virtual Route (VR) transmission queue for that session is congested. Specifically, session level pacing responses are intercepted at the entry node to exercise *network access* flow control from the terminal into the high-level network [16]. Thus, session pacing may be viewed as a hybrid form of transport level flow control, which is obtained by concatenating a network access level segment (from the terminal to the high-level network node) and an entry-to-exit level segment (controlled by virtual route pacing).

VII. CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

In this paper we have proposed a taxonomy of flow control mechanisms based on a multilevel structure. We have defined four levels of flow control and have shown how these levels are actually embedded into corresponding levels of protocols. To the extent that these levels can be independently defined, the analysis, design evaluation and comparison of flow control schemes is greatly simplified, since any complex control structure can be decomposed into smaller modules, and each module individually analyzed. The overall performance is then obtained by studying the interaction of the various modules.

Recent advances in queueing theory have led to reasonable success in the modeling and analysis of individual levels of flow control. We have reported on several performance results, and have used such results to compare different schemes.

In real life, however, some control structures defy the simple, hierarchical representation here proposed, and seem to combine two or more levels into hybrid solutions (see Fig. 10). This is particularly common in homogeneous networks (e.g., SNA) in which a single manufacturer is responsible for the implementation of both DCE and DTE equipment and, therefore, has more freedom in the design of the various flow control levels.

The existence of multiple levels of flow control and the possible integration of some of these into hybrid arrangements immediately brings up a very critical issue in flow control which requires further study, namely, the *interaction between levels*. Given that we understand the throughput and delay implications of each specific level of flow control, we still have to study the combined effect when these levels are operating simultaneously in the network. For instance, network experience seems to indicate that a network equipped with a very conservative hop level flow control, such as the SBP scheme in GMDNET or the VC-HL scheme in Tymnet, does not require strong network access or ETE flow control schemes since network congestion situations are immediately reported back to the entry node by back pressure through the hop level [36]. This type of issue can be fully investigated only by developing models which include multiple levels of flow control. An interesting example in this direction was the combined isarithmic and entry to exit flow control model presented in [47]. More research is required in this area.

Hybrid packet and circuit networks are now emerging as a solution to multimode (voice and data; batch and interactive) user requirements [11]. These networks must be equipped with novel flow control mechanisms. In fact, if the network were to apply conventional flow control schemes to the packet switched (P/S) component only, leaving the circuit switch (C/S) component uncontrolled, then the C/S component would very likely capture the entire network bandwidth during peak hours. This may not cause congestion, since the C/S protocol is not as congestion prone as the P/S protocol, but it certainly creates unfairness. Some form of flow control on C/S traffic which is sensitive to the relative P/S load is therefore required.

The *integration of voice and data* requirements in packet switched networks has been vigorously advocated in recent years on grounds of improved efficiency and reduced cost [14]. Unfortunately, little attention has been given to the fact that integrated networks require a complete redesign of the conventional flow control schemes since voice traffic cannot be buffered and delayed in case of congestion. Priorities are of help only if the voice traffic is a small fraction of the total traffic. For the general case, new flow control techniques must be developed for voice. These techniques should be *preventive* in nature, i.e., they should block calls before congestion occurs, rather than *detecting* congestion and then attempting to *recover* from it, as is the case for most of the conventional flow control schemes for data [10], [31].

Routing and flow control procedures have traditionally been developed independently in packet networks, under the assumption that flow control must keep excess traffic out of the network, and routing must struggle to efficiently transport to destination whatever traffic was permitted into the network by the flow control scheme. It seems, however, that routing and flow control can be brought together into useful cooperation in virtual circuit networks, where a path must be selected before data transfer on a user connection begins [12]. In this case, the routing algorithm can be invoked first to determine whether a path of sufficient residual bandwidth is available. If no path is available, the virtual circuit connection is blocked immediately at the entry node by the network access flow control level, thus *preventing* congestion rather than allowing it to occur and then attempting to *recover* from it. A combined routing and flow control strategy is implemented in Tymnet [39].

Challenging flow control problems exist in *multiaccess broadcast networks*. In single hop multiaccess systems, congestion prevention and stability mechanisms are well understood, and are usually directly embedded in the channel access protocol [46]. In distributed, multihop, multiaccess systems (e.g., multihop ground radio networks), congestion prevention becomes a very hard problem because of the interaction between buffer and channel congestion. Conventional flow control schemes used in hardwired nets cannot be directly applied. In particular, the hop level flow control should be revised to combine the buffer allocation strategy with the retransmission control strategy. Some pioneering work in this direction is reported in [2], [48], [43].

Finally, growing user demands require the *interconnection* of networks which may implement different flow control policies and which may even be built on different media (e.g., satellite, radio, cable, or optical fiber). These networks are interconnected by gateways which provide for internet routing and flow control, as well as for protocol conversion between two adjacent networks [44], [6]. It appears that a new level of flow control must therefore be defined in our hierarchy, namely, the gateway-to-gateway level. This level should be designed to prevent the congestion of gateways along the path, and should be supported by explicit gateway-to-gateway protocols for the exchange of status information. The status information should include buffer occupancy at the gateway, and load conditions in the adjacent networks, and could probably be exploited also for gateway routing. Functionally, the gateway-to-gateway protocol is positioned between the entry-to-exit protocol and the transport protocol hierarchy in Fig. 9. All the other levels remain unchanged. The actual implementation of the gateway-to-gateway flow control will be dependent on the internet protocol used. If the CCITT X.75 Recommendation, which is an extension of the X.25 virtual circuit concept to internet connections [45] is adopted, the gateway-to-gateway flow control will be virtual-circuit oriented, and will be exercised on a connection-by-connection basis. Alternatively, datagram-oriented gateway level flow control schemes can also be implemented.

The design of efficient gateway flow control schemes is very challenging. It requires *vertical* consistency between the gateway level and all the other levels implemented in each individual network as well as *horizontal* consistency across the various networks on the internet path. Specifically, the gateway level flow control must be able to balance loads between extremely diverse network environments such as point-to-point, satellite, cable, and ground radio. These design requirements further emphasize the need for continuing research in *multilevel* flow control models in order to understand the vertical interactions between the various levels in the hierarchy, as well as the horizontal interactions between the various segments of a flow control chain along an internet path.

In summary, we have presented a framework for the study of flow control, showing that flow control mechanisms have advanced somewhat beyond simply being "a bag of tricks" [34], and indeed can be conceptually organized into a useful and well-structured system of controls. This structure is extremely helpful in the survey and comparison of existing flow control implementations, as well as in the development of flow control models. In particular, complex control systems can be (and should be) decomposed into smaller modules, thus simplifying the analysis of each module as well as the analysis of interactions between different modules. Furthermore, the proposed flow control structure is sufficiently flexible to permit extensions in response to new networking technologies and applications.

Although our focus has been on flow control models and performance criteria, we expect that the proposed structure will prove to be useful also for the actual implementation of flow control techniques. One must be aware, of course, of the fact that in actual networks, it is not always possible to develop and update flow controls in a well structured fashion. The designer, in fact, is usually confronted with a number of constraints imposed by the preexisting protocol structure (in which flow control mechanisms must be embedded) and by limited storage and processing resources. The designer must therefore avoid overburdening the switch with overly sophisticated flow control mechanisms, and creating inconsistencies and possibly deadlocks. These constraints, together with the fact that flow control is a distributed multilevel control function that cannot be confined to a well-defined modular "black box," make flow control design a very hard task. It is our strong opinion, however, that the only way to prevent flow control implementations from degrading to the state of an uncontrollable "bag of tricks" is to identify an underlying structure in the early stage of flow control design, and to continuously verify this structure during the various updates of protocols and flow control procedures.

Indeed, it is important that one be able to subject a proposed flow control algorithm to various tests of correctness, consistency and proper termination [33], [49]. This is, in general, a very difficult task whose solution requires advances in the frontier of computer science. Unfortunately, since it is relatively difficult to create efficient, deadlock-free, flow control algorithms, we cannot totally ignore this need for verification. Moreover, many difficulties with flow control procedures often arise due to errors in the detailed implementation of otherwise correct algorithms. Consequently, it is important that a modular approach to flow control design be taken, that the code itself be confined to isolated portions of the network operating system (rather than sprinkled through thousands of lines of code) and that the mechanisms be simple enough to be understood and tested via simple procedures.

REFERENCES

[1] V. Ahuja, "Routing and flow control in systems network architecture," *IBM Syst. J.*, vol. 18, no. 2, pp. 298–314, 1979.

[2] G. Akavia and L. Kleinrock, "Performance tradeoff distributed packet-switching communication networks," Dep. Comput. Sci., School of Eng. Appl. Sci., Univ. of California, Los Angeles, Tech. Rep. UCLA-ENG-7942, Sept. 1979.

[3] P. Brinch-Hansen, *Operating System Principles*. Englewood Cliffs, Prentice-Hall, 1973.

[4] S. Carr *et al.*, "Host/host protocol in the ARPA network," in *Proc. Spring Joint Comput. Conf.*, 1970, pp. 589–597.

[5] V. G. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Trans. Commun.*, vol. COM-22, May 1974.

[6] V. G. Cerf, "DARPA activities in packet network interconnection," in *Interlinking of Computer Networks* (NATO Advanced Study Inst. Series). Reidel.

[7] A. Danet *et al.*, "The French public packet switching service: The Transpac network," in *Proc. Conf. Comp. Commun.*, Toronto, Ont., Canada, Aug. 1976.

[8] D. W. Davies, "The control of congestion in packet-switching networks," *IEEE Trans. Commun.*, vol. COM-20, June 1972.

[9] H. C. Folts, "International standards in computer communications," in *Proc. Nat. Telecommun. Conf.*, Nov. 1979, pp. 59.5.1–59.5.5.

[10] J. Forgie and A. Nemeth, "An efficient packetized voice/data network using statistical flow control," in *Proc. Int. Conf. Commun.* Chicago, IL, June 1977.

[11] M. Gerla and G. DeStasio, "Integration of packet and circuit transport protocols in the TRAN data network," in *Proc. Comput. Network Symp.*, Liege, Belgium, Feb. 1978.

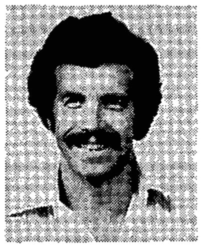
[12] M. Gerla, "Routing and flow control in virtual circuit computer networks," in *Proc. INFO II Int. Conf.*, July 1979.

[13] A. Giessler *et al.*, "Free buffer allocation—An investigation by simulation," *Comput. Networks*, vol. 2, pp. 191–208, 1978.

[14] I. Gitman and H. Frank, "Economic analysis of integrated voice and data networks," *Proc. IEEE*, pp. 1549–1570, Nov. 1978.

[15] J. P. Gray, "Network services in systems network architecture," *IEEE Trans. Commun.*, vol. COM-25, pp. 104–116, Jan. 1977.
[16] J. P. Gray and T. B. McNeill, "SNA multiple-system networking," *IBM Syst. J.*, vol. 18, no. 2, 1979.
[17] P. E. Green, "An introduction to network architectures and protocols," *IBM Syst. J.* no. 2, 1979, reprinted in this issue, pp. 413–424.
[18] M. Irland, "Buffer management in a packet switch," *IEEE Trans. Commun.*, vol. COM-26, pp. 328–337, Mar. 1978.
[19] R. E. Kahn and W. R. Crowther, "A study of the ARPA computer network design and performance," Bolt Beranek and Newman, Inc., Tech. Rep. 2161, Aug. 1971.
[20] F. Kamoun, "Design considerations for large computer communications networks," Ph.D. dissertation, Univ. of California, Los Angeles, Eng. Rep. 7642, Apr. 1976.
[21] F. Kamoun, "A drop and throttle flow control (DTFC) policy for computer networks," presented at the 9th Int. Teletraffic Congr., Spain, Oct. 1979.
[22] P. Kermani and L. Kleinrock, "Dynamic flow control in store and forward computer networks," *IEEE Trans. Commun.*, vol. COM-27, Feb. 1979.
[23] L. Kleinrock, *Queueing Systems: Volume II, Computer Applications*. New York: Wiley-Interscience, 1976.
[24] ——, "On flow control in computer networks," in *Proc. Int. Conf. Commun.*, June 1978.
[25] ——, "Power and deterministic rules of thumb for probabilistic problems in computer communications," in *Proc. Int. Conf. Commun.*, June 1979.
[26] L. Kleinrock and P. Kermani, "Static flow control in store and forward computer networks," *IEEE Trans. Commun.*, vol. COM-27, Feb. 1979.
[27] S. Lam and M. Reiser, "Congestion control of store and forward networks by buffer input limits," in *Proc. Nat. Telecommun. Conf.*, Los Angeles, CA, Dec. 1977.
[28] R. Magoon and D. Twyer, "Flow and congestion control in SL-10 networks," in *Proc. Int. Symp. Flow Control Comput. Networks*, Versailles, France, Feb. 1979.
[29] J. C. Majithia *et al.*, "Experiments in congestion control techniques," in *Proc. Int. Symp. Flow Control Comput. Networks*, Versailles, France, Feb. 1979.
[30] J. M. McQuillan *et al.*, "Improvements in the design and performance of the ARPA network," in *Proc. Fall Joint Comput. Conf.*, 1972.
[31] W. E. Naylor, "Stream traffic communication in packet-switched networks," Ph.D. dissertation, Dep. Comput. Sci., School Eng. Appl. Sci., Univ. of California, Los Angeles, Sept. 1977.
[32] M. Pennotti and M. Schwartz, "Congestion control in store and forward tandem links," *IEEE Trans. Commun.*, Dec. 1975.
[33] J. Postel, "A graph model analysis of computer communications protocols," Ph.D. dissertation, Univ. of California, Los Angeles, Jan. 1974.
[34] L. Pouzin, "Flow control in data networks—Methods and tools," in *Proc. Int. Conf. Comp. Commun.*, Toronto, Ont. Canada, Aug. 1976.
[35] W. L. Price, "Data network simulation experiments at the National Physical Laboratory," *Comput. Networks*, vol. 1, 1977.
[36] W. L. Price, "A review of the flow control aspects of the network simulation studies at the National Physical Laboratory," in *Proc. Int. Symp. Flow Control in Comput. Networks*, Versailles, France, Feb. 1979.
[37] E. Raubold, and J. Haenle, "A method of deadlock-free resource allocation and flow control in packet networks," in *Proc. Int. Conf. Comp. Commun.*, Toronto Ont., Canada, Aug. 1976.
[38] M. Reiser, "A queueing network analysis of computer communication networks with window flow control," *IEEE Trans. Commun.*, pp. 1199–1209, Aug. 1979.
[39] J. Rinde, "Routing and control in a centrally directed network," in *Proc. Nat. Comput. Conf.*, Dallas, TX, June 1977.
[40] J. Rinde and A. Caisse, "Passive flow control techniques for distributed networks," in *Proc. Int. Symp. Comput. Networks*, Versailles, France, Feb. 1979.
[41] M. Schwartz, and S. Saad, "Analysis of congestion control techniques in computer communications networks," in *Proc. Int. Symp. Comput. Networks*, Versailles, France, Feb. 1979.
[42] J. M. Simon and A. Danet, "Controle des ressources et principes du routage dans le reseau TRANSPAC," in *Proc. Int. Symp. Comput. Networks*, Versailles, France, Feb. 1979.
[43] J. Silvester "On spatial capacity of packet radio networks," Ph.D. dissertation, Dep. Comput. Sci., School Eng. Appl. Sci., Univ. of California, Los Angeles, Mar. 1980.
[44] A. C. Sunshine, "Interconnection of computer networks," *Comput. Networks*, vol. 1, 1977.
[45] A. C. Sunshine, "Transport protocols for computer networks," in *Protocols and Techniques for Data Communications Networks*, F. Kuo, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1980.
[46] F. Tobagi, "Multiaccess protocols in packet communication systems," this issue, pp. 468–488.
[47] J. W. Wong, and M. S. Unsoy, "Analysis of flow control in switched data networks," in *Proc. Int. Fed. Inf. Processing Soc. Conf.*, Aug. 1977.
[48] Y. Yemini and L. Kleinrock, "On a general rule for access control or, silence is golden...," in *Proc. Int. Symp. Flow Control Comput. Networks*, Versailles, France, Feb. 1979.
[49] P. Zafiropulo, "A new approach to protocol validation," in *Proc. Int. Conf. Commun.* June, 1977.
[50] H. Zimmermann, "The Cyclades end-to-end protocol," in *Proc. 4th Data Commun. Symp.*, Quebec, P. Q., Canada, Oct. 1975, pp. 7:21–26.
[51] Schwartz and Stern, "Routing algorithms: A comparative survey," this issue, pp. 539–552.

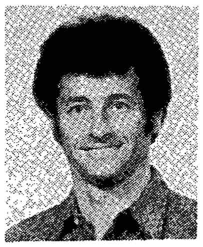
★



Mario Gerla (M'75) received a graduate degree in electrical engineering from the Politecnico of Milan, Milan, Italy, in 1966, and the Ph.D. degree in engineering from the University of California, Los Angeles, in 1973.

He came to UCLA with a Fulbright scholarship in 1969, and worked on the ARPA network project as a Research Assistant. In January 1973, he joined Network Analysis Corporation, Glen Cove, NY, where as Manager of Computer Networking, he was involved in several computer network design projects for both government and industry. Since July 1977, he has been on the faculty of the Department of Computer Science at UCLA. His research interests are in the areas of design, control, and performance evaluation of data communications networks and distributed processing systems.

★



Leonard Kleinrock (S'55–M'64–SM'71–F'73) received the B.E.E. degree from the City College of New York, New York, NY, in 1957, and the M.S.E.E. and Ph.D.E.E. degrees from Massachusetts Institute of Technology, Cambridge, in 1959 and 1963, respectively.

In 1963, he joined the faculty of the School of Engineering and Applied Science, University of California, Los Angeles, where he is now Professor of Computer Science. His research spans the fields of computer networks, computer systems modeling and analysis, queueing theory, and resource sharing and allocation in general. At UCLA, he directs a large group in advanced teleprocessing systems and computer networks. He is the author of three major books in the field of computer networks: *Communication Nets: Stochastic Message Flow and Delay* (New York: McGraw-Hill, 1964; also New York: Dover, 1972); *Queueing Systems, Vol. I. Theory* (New York: Wiley-Interscience, 1975); and *Queueing Systems, Vol. II: Computer Applications* (New York: Wiley-Interscience, 1976). He has published over 100 articles and contributed to several books. He serves as consultant for many domestic and foreign corporations and governments, and he is a referee for numerous scholarly publications and a book reviewer for several publishers.

Dr. Kleinrock is a Guggenheim Fellow and has received various outstanding teacher and best paper awards, including the 1976 Lanchester prize for the outstanding paper in operations research, and the ICC 1978 prizewinning paper award.