Prevention of Deadlocks and Livelocks in Lossless Backpressured Packet Networks

Mark Karol, *Fellow, IEEE*, S. Jamaloddin Golestani, *Fellow, IEEE*, and David Lee, *Fellow, IEEE*

***Abstract*—No packets will be dropped inside a packet network, even when congestion builds up, if congested nodes send backpressure feedback to neighboring nodes, informing them of unavailability of buffering capacity—stopping them from forwarding more packets until enough buffer becomes available. While there are potential advantages in backpressured networks that do not allow packet dropping, such networks are susceptible to a condition known as deadlock in which throughput of the network or part of the network goes to zero (i.e., no packets are transmitted). In this paper, we describe a simple, lossless method of preventing deadlocks and livelocks in backpressured packet networks. In contrast with prior approaches, our proposed technique does not introduce any packet losses, does not corrupt packet sequence, and does not require any changes to packet headers. It represents a new networking paradigm in which internal network losses are avoided (thereby simplifying the design of other network protocols) and internal network delays are bounded.**

***Index Terms*—Backpressure, bounded delay, congestion control, deadlock prevention, livelock prevention, lossless networks.**

I. Introduction

CONGESTION occurs in packet networks when the demand exceeds the availability of network resources, leading to lower throughputs and higher delays. If congestion is not properly controlled, some sessions transported by the network may not meet their quality-of-service (QoS) requirements. When congestion builds up in a packet network, two general approaches are possible to cope with the shortage of buffer space. One approach is to drop incoming packets for which buffer is not available and to rely on the end-to-end protocols for the recovery of lost packets. The alternative approach is to insist that no packets should be dropped inside a packet network, even when congestion builds up. This goal may be accomplished by, for example, having the congested nodes send *backpressure* feedback to neighboring nodes, informing them of unavailability of buffering capacity and in effect stopping them from forwarding more packets until enough buffer becomes available. Lossless networks that employ backpressure signals on a hop-by-hop basis may be referred to as *backpressured* networks. An example of networking based on backpressure signals can be seen in the PAUSE signals of gigabit ethernet (IEEE 802.3z) technology.

Manuscript received October 9, 2000; revised May 21, 2002; approved by IEEE/ACM Transactions on Networking Editor M. Ammar.

M. Karol is with Avaya Inc., Lincroft, NJ 07738-1526 USA (e-mail: mk@avaya.com).

S. J. Golestani and D. Lee are with Lucent Technologies, Murray Hill, NJ 07974 USA (e-mail: golestani@lucent.com; leedavid@lucent.com).

Digital Object Identifier 10.1109/TNET.2003.820434

The dominant approach in today's networking technology is based on dropping packets at the congested nodes and relying on end-to-end recovery, as seen for example in TCP congestion control. However, in the LAN context, simulation results show that hop-by-hop backpressure is often better than TCP for dealing with short-lived congestion [1]. TCP uses packet drops as an indication of congestion and requires sufficient levels of loss in order to be an effective control.

While there are potential advantages in backpressured networks that do not allow packet dropping, such networks are susceptible to a condition known as *deadlock* in which throughput of the network or part of the network goes to zero (i.e., no packets are transmitted). There is entire stoppage because, for example, one network process, having resources required by another, refuses or neglects to release them, causing the other process to wait indefinitely. Many different types of deadlocks have been identified and studied, along with methods of preventing them (see, e.g., [2]–[5]).

In addition to deadlocks, a more subtle condition known as *livelock* can occur in backpressured networks in which, although communication in the network or part of the network is not halted, one or more individual packets are never transmitted. It is an extreme example of "unfairness," in which packets are "stuck" as they wait indefinitely in a queue while other packets (including new arrivals) are continually served (because, for instance, they have a higher service priority). Sometimes, in an attempt to design a deadlock-prevention algorithm for a backpressured network, it is easy to end up with a solution that prevents deadlocks but creates livelocks.

In contrast with prior approaches, our proposed technique prevents deadlocks and livelocks in backpressured networks without introducing any packet losses, without corrupting packet sequence, without relying on elaborate network-wide coordinations requiring multihop control messages, and without requiring any change to packet headers. These last two points are particularly important, since we insisted in our design that no changes be necessary to existing ethernet standards. This technique represents a *new networking paradigm* in which internal network losses are avoided (thereby simplifying the design of other network protocols) and internal network delays are bounded.

In Section II, we review some existing deadlock prevention techniques. In Section III, we present a new, simple, lossless method of preventing deadlocks and livelocks in packet networks. We describe the protocol and include proofs that it is deadlock- and livelock-free. In Section IV, we discuss some remaining important implementation issues, including the incorporation of nonzero propagation delays in the protocol and the handling of route updates. Finally, in Section V, we conclude with some alternatives and extensions to the basic protocol.

II. Deadlock Prevention in Networks

One simple way to deal with deadlocks is to just start dropping packets once a deadlock has occurred or is "about to occur" (i.e., as congestion increases). For certain types of packets (for example, "real-time" traffic and traffic that can permit packet loss), this approach works fine. However, packets that *must* eventually reach their destinations will need to be retransmitted if they are dropped. Naturally, the impact of such retransmissions on total end-to-end delay, including interactions with higher layer protocols (such as TCP), needs to be considered. Moreover, retransmission of packets might even cause the "deadlock condition" to redevelop.

Alternatively, if peak amounts of bandwidth and buffers are available and are dedicated for all network traffic flows, then buffers do not overflow and deadlocks are not created. This approach, however, is typically too wasteful of network resources and requires stringent admission control procedures.

Deadlocks can also be avoided by assigning directions to the links such that cycles are avoided, using, for example, up/down routing on a spanning tree [6]. However, the selected paths (to create the spanning tree) are generally not the shortest and links near the root of the spanning tree become bottlenecks, limiting the network throughput.

Another way to avoid cycles in the network is to split each physical link into a number of virtual channels, each with its own queue and backpressure protocol [7], [8]. Bandwidth is shared among the virtual channels. Layers of acyclic virtual networks and deadlock-free routes are created using the virtual channels. However, as the number of virtual channels increases, the scheduling becomes more complicated. Also, a method of associating individual packets with particular virtual channels is required.

Finally, more sophisticated buffer allocation strategies (structured buffer pools) can be used to prevent deadlocks. For instance, extra buffers can be allocated for packets that have higher "priority" because they have traveled greater distances (e.g., number of hops) in the network (or are closer to their destinations). In other words, using "distance information" in the packet headers, buffer space is reserved at each network node according to the distance traveled through the network from a source (i.e., the number of hops)—see [9, sec. 6.1.2 and fig. 6.3].

There are several problems with these prior deadlock prevention techniques. First and most important for the gigabit ethernet scenario, they may not be compatible with the IEEE 802.3z standard. Many prior approaches require packet headers to include the "distance information" (such as, for example, the packet's hop count). There is no such provision for including distance information in the header of IEEE 802.3z packets. Alternatively, some other ("nonstandard") way would need to be employed for transferring the "distance" information downstream to the next node.

A second problem with prior approaches is that although deadlocks are prevented, the end-to-end packet sequence may not be preserved. This may cause problems for some sessions that expect to receive packets in sequence, and perhaps can deal with "missing" (i.e., dropped) packets better than "out-of-sequence" packets. A third problem with prior approaches is that although they resolve the deadlock problem, some do not eliminate the possibility of livelock. In contrast, our protocol preserves packet sequence and avoids livelocks while it prevents deadlocks.

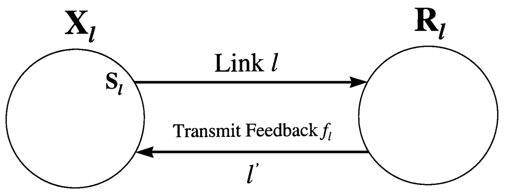


Fig. 1. Link l from X_l to R_l with scheduling algorithm $\mathcal{S}_\ell$.

Finally, a fourth problem with some prior approaches is that a method of determining the current "distance to destination" information must be available in the network. This gets particularly challenging when we account for the possibility of network reconfigurations and routing table updates, which can change the source-to-destination distances.

III. Selective Backpressure Protocol for Deadlock Prevention

Consider a packet network composed of one-way communication links ℓ. For each link ℓ, let us denote the sending node by X_ℓ, the receiving node by R_ℓ, and the communication link going in the reverse direction, from R_ℓ to X_ℓ, by ℓ' (see Fig. 1).[1]

Let $\mathcal{S}_\ell$ denote the scheduling algorithm of link ℓ, i.e., the algorithm employed at node X_ℓ to select the next packet from those buffered at X_ℓ for transmission over ℓ. The scheduling algorithm $\mathcal{S}_\ell$ may base its selection on a number of factors, including packets' order of arrival, service priorities, service deadlines, and fairness considerations. As described earlier, packet losses in the network may be avoided by employing a backpressure congestion control mechanism for each link ℓ. In this mechanism, before the buffer at the receiving node R_ℓ of a link ℓ overflows, a *stop* feedback is sent to the sending node X_ℓ, over the reverse link ℓ'. However, as described in Section II, this backpressure mechanism can lead to the occurrence of deadlocks and livelocks in the network.

In this section, we describe a simple protocol **P** that prevents the occurrence of deadlocks while maintaining the benefits of a lossless backpressure congestion control. In addition, livelocks will not occur if the scheduling algorithms $\mathcal{S}_\ell$ are *well-behaved* in the sense that they do not continually neglect transmission of any particular packet (as could happen to a low-priority packet, for instance, with a strict priority-based scheduling algorithm). The avoidance of deadlocks and livelocks is accomplished without making any changes to packet headers, such as attaching a *hop counter*, as required by the prior approaches described in Section II. The proposed technique makes use of only the *Destination Address* (or alternative routing information such as the virtual circuit number) that already exists in each packet

[1]Each node in the network is, at the same time, both a sending node and a receiving node for its respective output and input links.

header. The format of a typical gigabit-ethernet packet, for example, contains the Destination Address but not a hop-counter field.

Our technique is quite general and can be used for networks with various types of routing and different transmission speeds and geographic spans. However, throughout this section, in order to focus on the essence of the protocol and not be sidetracked by secondary issues, we make a few assumptions with respect to routing, packet forwarding, and propagation delays in the network. First, we assume that the network routing is static, thereby ignoring routing updates that may take place in response to topology changes or traffic conditions. Second, we assume that packet forwarding in the network is *destination-based*, i.e., the next hop used to forward a packet from one node to another is based solely on the packet's destination. With destination-based packet forwarding, all the (original and transit) traffic going from a given node to a given destination follow the same network path. Finally, we assume in this section that all network links have zero propagation delays. These simplifying assumptions will all be relaxed in Section IV, where we will present variations of the basic protocol developed here, in order to accommodate nonzero delays, adaptive routing, and packet forwarding methods that utilize information other than a packet's destination (such as in ATM, the virtual circuit number).

A. The Protocol

The simple and new technique that we describe in this section can be viewed as a *selective* backpressure mechanism (in contrast with nondiscriminating backpressure). Under normal, uncongested conditions, all packets waiting at a sending node X_ℓ, to be sent over link ℓ, are eligible for transmission and may be selected by the link scheduling algorithm $\mathcal{S}_\ell$ (see Fig. 1). However, as the buffer at the receiving node R_ℓ gets congested, e.g., fills up close to its capacity, the proposed protocol **P** gradually restricts the set of packets that are *eligible* for transmission (i.e., those packets that may be selected by the scheduling algorithm $\mathcal{S}_\ell$ to be sent over link ℓ). This is accomplished by sending a *transmit feedback* parameter f_ℓ, to be discussed shortly, over the communication link ℓ' in the reverse direction, from R_ℓ to X_ℓ (see Fig. 1). As the congestion at node R_ℓ subsides, the restriction placed on the transmission of packets over ℓ is gradually lifted by designating more packets as *eligible* using a new transmit feedback f_ℓ. Values of the transmit feedback f_ℓ are selected such that there will be room in the buffer at the receiving node R_ℓ to store packets subsequently received from the sending node X_ℓ. In the absence of transmission or processing errors, this will guarantee that packet transmission in the network is loss free.

The selective backpressure protocol **P** that we propose here is general and applicable to different networks with various routing and scheduling protocols. In this paper, we only specify the basic requirements for the protocol and prove its general properties, i.e., freedom from loss, deadlock and livelock. While the protocol may be used in networks with different scheduling algorithms and may be implemented using a variety of buffer management schemes (to address issues such as efficiency and fairness), its correctness is guaranteed so long as the basic requirements are satisfied. On the other hand, we shall discuss some scheduling and buffer management schemes as case studies to give more insight into our protocol.

Let D denote the maximum number of hops in any legitimate network route (or an upper bound on the number of hops if the maximum is, a priori, unknown). D depends on the network topology and routing protocol. For instance, if shortest path routing is used, then D is the diameter of the network. We associate with each packet p buffered at a node an integer value between 0 and D, inclusive; we call that assigned value the *level* of p and denote it as λ_p. It is "local information"; when a packet is forwarded in the network from one node to another, no information about the level that was assigned to the packet in the first (sending) node is carried along with it. For instance, packet levels are *not* included in the packet headers (in contrast with, for example, prior deadlock-prevention schemes that carry hop counts in packet headers). To carry such information would require a change in existing ethernet standards. Nonetheless, as we will see, protocol **P** allows the receiving node to infer some partial information about the level that the packet held in the previous node. Typically, the level assigned to the packet in the new (receiving) node is different than its previous level.

Like packet levels, the transmit feedback f_ℓ also assumes an integer value between 0 and D, inclusive. The eligibility of packets for transmission over each link ℓ is determined by the value of the corresponding transmit feedback f_ℓ in accordance with the following rule.

Transmit Eligibility Rule: A packet p waiting at node X_ℓ is *eligible* to be picked up by the scheduler of link ℓ for transmission over ℓ, if its current level λ_p satisfies

$$\lambda_p \geq f_\ell \tag{1}$$

where f_ℓ is the most recent transmit feedback received by X_ℓ from the receiving node R_ℓ.

It follows from this rule that, when $f_\ell = 0$, all packets are eligible for transmission over ℓ. As f_ℓ increases, the protocol becomes gradually more restrictive and fewer and fewer packets are considered eligible for transmission over ℓ.

The assumption of zero propagation delays for network links that is made in this section implies that a transmit feedback f_ℓ sent by R_ℓ reaches X_ℓ instantly. In Section IV-C, we will discuss necessary modifications in the protocol to accommodate real network scenarios where this condition is not met.

Next, we describe how the level of a packet arriving at a node is determined. Let packet p arrive to node R_ℓ over link ℓ and assume that f_ℓ is the most recent transmit feedback sent to X_ℓ. It follows that the level of p prior to transmission from the previous node (X_ℓ) must have been f_ℓ or larger. To guarantee freedom of deadlock in the network, it suffices to assign level $1 + f_\ell$ to packet p (as well as follow the buffer management and feedback rules described below). However, this simple approach can lead to the assignment of different levels, at a given node, to packets of the same session, which in turn can result in misordering of these packets when they are forwarded to the next downstream node.

To avoid misordering of packets belonging to the same session, for networks with *destination-based* packet forwarding that are considered in the present section, we have adopted the following principle in the design of protocol **P**: at each node and at each point of time, all buffered packets that have a common destination should have the same level so that all will be eligible/ineligible at the same times, and therefore selected for transmission in the correct order. We accomplish this principle, at each node, by lifting the level of all packets with a common destination to the highest level among them (which also potentially increases their opportunities to be eligible for transmission). With this modification, it is more appropriate to view the level assignments at a node as being performed on a per-destination, rather than per-packet, basis. In accordance with this viewpoint, at a given node, let us denote the level associated with a destination d, by λ^d. At any point of time, the level λ_p of each packet p buffered at node n is equal to the level λ^d associated with the corresponding destination d, at that time.

Finally, the new level λ^d (at R_ℓ) is determined in accordance with the following rule.

Level Assignment Rules:

1) At each node, initially set $\lambda^d = 0$ for all destinations d.
2) When a packet p with destination d arrives from another network node over some link ℓ, the level associated with d is updated as

$$\lambda^d \longleftarrow \max\left(\lambda^d,\ 1 + f_\ell\right) \tag{2}$$

where f_ℓ is the value of the most recent transmit feedback sent over the reverse link ℓ'.
3) When a packet p with destination d enters the network at node n (over some network access link) the destination level λ^d does not change.

Every node n maintains a list of all destinations d that it encounters, along with the associated level λ^d. We refer to this list as the *level table* of node n. Initially, there are no entries in the level table. By default, the level of any destination not included in the level table is zero. Accordingly, any destination which has a level equal to zero may be eliminated from the level table. When all buffered packets with destination d have left node n, it is permissible (but not necessary) to eliminate destination d from the level table at n. In other words, it is only necessary to keep values in the level table for destinations of currently buffered packets. This limits the size of the level table; the level table can be quite small (compared with the number of all possible destinations). Note that such elimination is equivalent to resetting λ^d to zero.

Before proceeding, we present a few important observations regarding the above rules.

- If all packets encountered by node n and destined for destination d enter the network at n, then λ^d is always equal to zero (at n) since it is never subjected to the update in (2). This means that packets arriving into the network at node n and destined for d will assume level zero at n, provided that n does not encounter any traffic destined for d that comes from another network node.
- When a packet arrives at node n from another network node, it will be assigned a level of at least 1, since the level associated with its destination will undergo the update in (2).

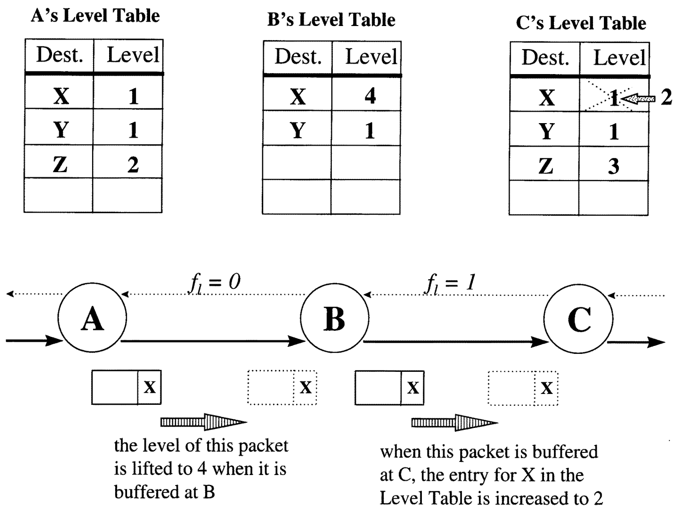

A's Level Table
Dest. | Level
X | 1
Y | 1
Z | 2
B's Level Table
Dest. | Level
X | 4
Y | 1
C's Level Table
Dest. | Level
X | 1
2
Y | 1
Z | 3
$f_l = 0$
$f_l = 1$
A
B
C
X
X
X
X
the level of this packet is lifted to 4 when it is buffered at B
when this packet is buffered at C, the entry for X in the Level Table is increased to 2

Fig. 2. Sample of levels and transmit feedback values.

- Updating according to the above rules will never result in a level larger than D. As will be shown later (Lemma 1 in Section III-B), by the time the level associated with a packet reaches D, the packet must have reached its destination (otherwise, an error has occurred and the packet can be dropped). Typically, packets' levels are less than D when they reach their destinations.
- Nodes do not need to keep track of the levels associated with each and every individual buffered packet. Each node only needs maintain a short level table listing the destinations for which packets are buffered at the node.
- It is only necessary to keep values in a level table for destinations of currently buffered packets. However, there are some potential benefits to maintaining the information in the level table for longer periods of time. Keeping the values longer (i.e., even after all packets to particular destinations have departed) maintains some history about those destinations. This increases the eligibility levels of later-arriving packets (perhaps already in transit) to those same destinations and thereby potentially increases their opportunities for transmission. The subject of level tables and their refreshment will be revisited in Section IV where we relax the current assumption of static routing and deal with route updates in the network.

To illustrate how levels are assigned, Fig. 2 shows a network example of various levels and transmit feedback values. Node A has permission to transmit a packet destined to node X because the level associated with X (i.e., "1") is not less than the transmit feedback parameter (i.e., "0") of the link to node B. When the packet reaches and is buffered at node B, the level of the packet is lifted to "4," which is the value associated with X in node B's level table. Similarly, node B has permission to transmit a packet destined to node X because the level associated with X (i.e., "4") is not less than the transmit feedback parameter (i.e., "1") of the link to node C. When the packet reaches and is buffered at node C, the entry for X in node C's level table is increased to "2" (which automatically lifts the level to "2" of any packets destined for X that are buffered at C). Notice in this

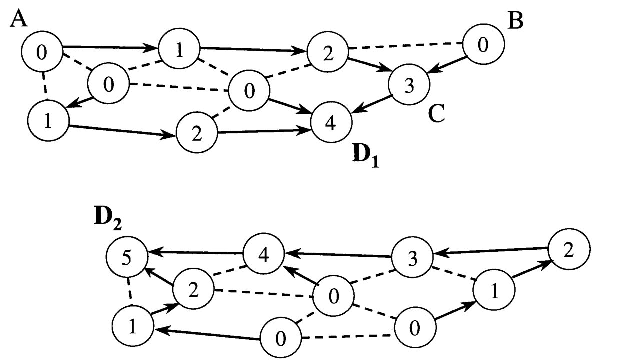


Fig. 3. Maximum possible values of destination levels.

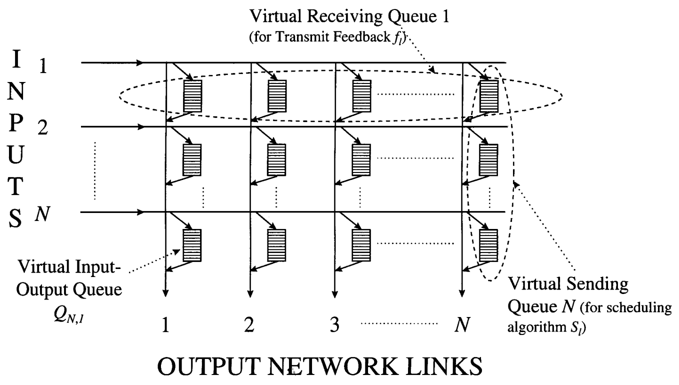

Virtual Receiving Queue 1
(for Transmit Feedback f_l)
INPUTS
1
2
N
Virtual Input-Output Queue $Q_{N,1}$
Virtual Sending Queue N (for scheduling algorithm S_l)
1
2
3
N
OUTPUT NETWORK LINKS

Fig. 4. General switch model.

simple example that the level of a packet destined for X went from "1" to "4" to "2" as it passed from A to B to C.

The example of Fig. 2 shows that a packet's level is quite "volatile" as it travels through a network; the level can even increase while it is buffered at a node (as at node C in Fig. 2). One of the few things that can be said about a packet's level is that it is *less than or equal to D minus the number of hops remaining to the packet's destination*. As an illustration, Fig. 3 shows a ten-node network with specific routing paths to reach destinations D_1 and D_2. Inside each node is an integer that indicates the maximum possible level of packets destined to D_1 (or D_2). Consider the D_1 example. At a leaf of the routing tree (e.g., nodes A and B), the level is always zero. A packet entering the network at node C, however, might be assigned a level as large as "3" because of the impact on C's level table from packets that entered the network at node A and that may have attained a level equal to 3 at node C. Likewise, the packets that enter at node B (with their levels initially set to "0") might have their levels lifted to "3" when they reach node C. Similar observations can be made with the set of paths that route packets to D_2 (which is labeled as node A in the D_1 example). In summary, a packet's level increases and decreases as a function of the topology, the distance the packet travels, the "recent history" of arrivals (throughout the network), and the state of network congestion.

Now that we have discussed: 1) the concepts of packet (or destination) levels; 2) the transmit feedback; and 3) the Transmit Eligibility Rule, we explain in the rest of this section how values for the transmit feedback are selected. First, though, we need to briefly describe the general switch architecture under consideration.

The proposed selective backpressure technique **P** can be used in networks with arbitrary switch configurations. So, in this paper, we consider a general switch model for each node of the network (see Fig. 4). A *virtual* input–output queue $Q_{i,j}$ is associated with each input–output pair (i, j), a *virtual* receiving queue is associated with each incoming network link, and a *virtual* sending queue is associated with each outgoing network link. The *receiving queue* associated with an incoming link ℓ is used for determining the transmit feedback f_ℓ, and the *sending queue* is used with its associated scheduling algorithm in determining the next packet to be transmitted—selecting from those packets that are eligible. Note that in the switch model adopted here, each packet is at the same time treated as belonging to both a receiving and a sending queue (corresponding to its input–output queue). Therefore, our earlier assumption that a scheduling algorithm $\mathcal{S}_\ell$ is well-behaved implies that (eligible) packets in each receiving queue (i.e., arriving from each of the input links) will eventually be selected for transmission over an output link.

This general switch model can be physically implemented in many ways, which is why we stress that the defined input–output, receiving, and sending queues are *virtual*. For instance, in a completely-shared-memory switch, all input–output, receiving, and sending queues are maintained in lists as packets arrive on various incoming links and depart on various outgoing links. In an input-buffered (output-buffered) switch, however, the receiving (sending) queue could be a physical buffer and the other queues would still be virtual entities that are maintained by, for example, keeping lists of packets destined for (coming from) various outgoing (incoming) links. Not shown in Fig. 4 are the buffers assigned for the traffic entering (and leaving) the network at a node (i.e., over some network access link). Protocol **P** does not specify anything regarding the admission policies and flow control on access links, nor does it specify anything regarding the sizing of the specific buffers assigned for receiving (transmitting) traffic over access links.

Now consider again an arbitrary *network* link ℓ and the receiving node R_ℓ. Let b^ℓ denote the (current) size of ℓ's Receiving Queue (we say "current" because some switch architectures may allow the size of the virtual queues to dynamically change over time). Also, let $\gamma_{\max}$ denote the maximum size of a packet in the network. Our core idea for deadlock prevention is to manage the receiving queue of each link ℓ and to set the value of the transmit feedback f_ℓ as described in the following paragraph.

The transmit feedback f_ℓ sent from a receiving node R_ℓ to a sending node X_ℓ is determined by first setting in the receiving queue thresholds B_i that limit the maximum amount of space for packets with levels less than or equal to i. At all times, all B_i buffer threshold constraints must be satisfied. This division is not a physical partitioning of the buffer space, but is only an allocation of space. Allocation typically occurs when the system is initialized. The receiving node R_ℓ thereafter monitors the levels of arriving and departing packets (and the "lifting" of levels of previously stored packets, which occurs so that all packets with a common destination have the same level), and

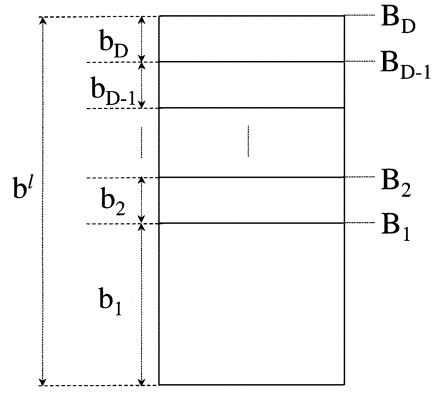


Fig. 5. Budget allocations of link ℓ's receiving queue.

thus keeps track of the total space in the receiving queue occupied by packets of various levels. The transmit feedback f_ℓ sent to the sending node X_ℓ represents the lowest level of packets (at X_ℓ) that the receiving queue could accept without violating any of the B_i buffer threshold constraints. In other words, the receiving queue has room to accept packets of level $(1 + f_\ell)$ or greater, without violating any of the buffer threshold constraints, but the receiving queue cannot accept packets of level f_ℓ or lower because it could possibly cause one or more of the buffer threshold constraints to be violated. In the remainder of this section, we present more details of the buffer management and the calculation of the transmit feedback f_ℓ.

As illustrated in Fig. 5, we divide the total buffer size b^ℓ into D parts b_i, $i = 1, 2, \ldots, D$, satisfying

$$b_i \geq \gamma_{\max} \tag{3}$$

and

$$b^\ell = \sum_{j=1}^{D} b_j \geq D \cdot \gamma_{\max}. \tag{4}$$

This division is not a physical partitioning of the buffer space; it is only an allocation of space.[2] We refer to b_i as the *buffer budget* of level i and require that a packet of level i be accepted into the buffer only if there is enough budget available for it at levels i or below. Let n_i, $i = 1, 2, \ldots, D$, denote the combined size of packets of level i that are stored in the receiving queue of link ℓ. The above requirement may be stated as

$$n_1 \leq b_1 \tag{5}$$

$$n_1 + n_2 \leq b_1 + b_2 \tag{6}$$

or more generally

$$\sum_{j=1}^{j=i} n_j \leq \sum_{j=1}^{j=i} b_j, \quad i = 1, 2, \ldots D. \tag{7}$$

Equivalently, we may define buffer threshold constraints $B_i = \sum_{j=1}^{j=i} b_j$ that limit the maximum buffer capacity that can be occupied by packets of level less than or equal to i. At all times, all B_i buffer threshold constraints must be satisfied [as in (7)]. Note that $B_D = b^\ell$, the size of the receiving queue.

[2]The buffer size b^ℓ is only weakly dependent on the maximum route length D. Most of the buffer space is in b_1—i.e., b_j is very small for $j > 1$—and the partitioning is "virtual." The buffer space set aside solely for higher level packets is only used when congestion occurs and a small amount of space is needed to prevent deadlocks/livelocks.

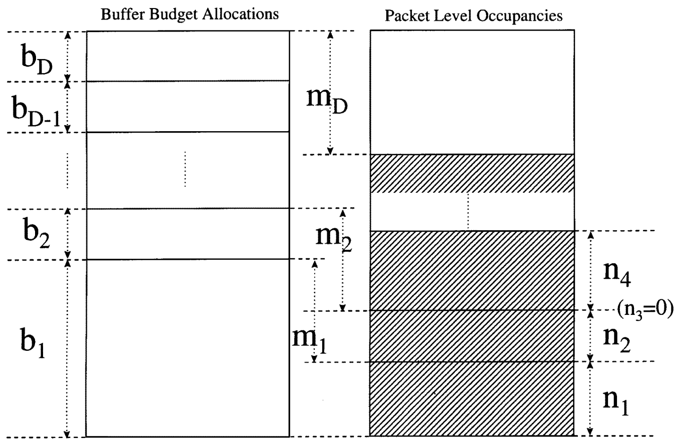


Fig. 6. Buffer management parameters—ℓ's receiving queue.

In order to observe the above requirements (to prevent deadlocks and livelocks), it is not necessary to physically partition the receiving queue into different segments. Instead, as illustrated in Fig. 6, the above requirement may be implemented using a set of buffer management parameters m_i defined as

$$m_i \triangleq \sum_{j=1}^{j=i} (b_j - n_j), \quad i = 1, 2, \ldots, D. \tag{8}$$

Equivalently, $m_i \triangleq B_i - \sum_{j=1}^{j=i} n_j$. m_i refers to that part of the combined buffer budget of levels $j \leq i$, which is not allocated to packets of levels $j \leq i$. This means that out of the combined buffer budget of levels $j \leq i$, a budget m_i is either allocated to packets of levels $j > i$ or not allocated to any packets at all. With this notation, m_D equals the total size of the vacant space in the buffer. Notice that since packets of level j can use the buffer budget of any level $k \leq j$, the term $b_j - n_j$ in (8) can be negative, for some j. However, m_i cannot be negative for any i since packets of levels $j \leq i$ cannot use the buffer budget of a level higher than i.

It follows from (8) that, when a new packet of length γ and level j is stored in the buffer (leaves the buffer), m_i must be decreased (increased) by γ for all $i \geq j$. Similarly, when the packet's level is lifted from j to k, then according to (8) m_i must be increased by γ for all levels i, $k > i \geq j$.

The above results also provide the guideline for choosing the transmit feedback f_ℓ to be sent over the reverse link ℓ' to the upstream node. Since the parameters m_i should always be nonnegative, the buffer can store a new packet of level j and arbitrary length, provided that currently $m_i \geq \gamma_{\max}$, for all $i \geq j$. On the other hand, when a packet arrives following the sending of the transmit feedback f_ℓ, the level assigned to it could be as low as $1 + f_\ell$. Since we would like to set the value of the transmit feedback f_ℓ as low as possible (to guarantee that the network will be free of livelocks and to increase the utilization of the link ℓ), we conclude that f_ℓ should be set to a level j such that $m_i \geq \gamma_{\max}$, for all $i \geq j + 1$, and $m_j < \gamma_{\max}$. It follows that f_ℓ should be set to the *highest* level j for which $m_j < \gamma_{\max}$. Accordingly, if $m_i \geq \gamma_{\max}$, for all $i = 1, 2, \ldots, D$, then we set $f_\ell = 0$, allowing the transmission of packets of any level, over link ℓ. Conversely, if $m_D < \gamma_{\max}$, then it follows that $f_\ell = D$. As we said before, if there is any packet of level D in the upstream node, it must be destined for that node itself. Therefore, all packets waiting to be sent over link ℓ must have a level less than D. It follows that, with the transmit feedback set to D, no packet is eligible to be sent over ℓ.

We now summarize the rules to be followed in protocol **P** for feedback setting of link ℓ and management of its receiving queue.

Buffer Management Rules:

1) When the receiving queue of link ℓ is empty, initiate

$$m_i = \sum_{j=1}^{j=i} b_j, \quad i = 1, 2, \ldots, D. \tag{9}$$

2) If a packet of length γ arrives and is buffered at level j, decrease m_i by γ, for $i \geq j$.
3) If a packet of length γ and level j leaves the buffer, increase m_i by γ, for $i \geq j$.
4) If a packet of length γ is lifted from level j to level $k > j$, increase m_i by γ, for all i, such that $k > i \geq j$.

Transmit Feedback Rules:

1) At the receiving end R_ℓ of each link ℓ, set the corresponding transmit feedback $f_\ell = j$, where j is the *largest* level for which $m_j < \gamma_{\max}$. If no such level exists, set $f_\ell = 0$.
2) Whenever f_ℓ changes, send an immediate feedback with the new value of f_ℓ to the transmitting node X_ℓ.

Finally, we point out that the order of packet transmissions over each link of the network can be altered as the result of applying the selective backpressure protocol **P**. In fact, by selectively designating packets as eligible for transmission, protocol **P** "interferes" with normal operation of scheduler $\mathcal{S}_\ell$ of each link ℓ. This is in contrast to the performance of a plain backpressure mechanism, which does not change the order of packet transmissions at the link level since either all packets waiting at a link are eligible for transmission or no packet may be sent at all.

We now proceed to formally state and prove the properties of the selective backpressure protocol **P**.

B. *Properties of Protocol* **P**

In order to present a formal statement of the properties of protocol **P**, we should first clarify what is meant by deadlock or livelock. The conditions of deadlock and livelock may be defined in a number of ways. Here, for the purpose of the present networking discussion, we use the following definitions for freedom from deadlock and livelock.

Definition 1: A network is defined to be *deadlock-free* if, given an arbitrary combination of packets sitting in its buffers, the delivery of each packet to its destination is guaranteed within a finite time, provided that there are no new packet arrivals to the network.

Definition 2: A network is defined to be *livelock-free* if, given an arbitrary combination of packets sitting in its buffers and an arbitrary pattern of new packet arrivals into the network, the delivery of each packet to its destination is guaranteed within a finite time.

While protocol **P** described in the foregoing section is sufficient to ensure deadlock-free operation for the network, freedom from livelocks is a more demanding condition. Unlike deadlocks, livelocks can result not only from network level problems related to interaction among different nodes, but also from scheduling features that can exist within an isolated node or link. To establish freedom of livelock at the network level we must first ensure that a similar condition, defined in the following, is satisfied at the level of individual links.

Definition 3: The *eligibility age* of a packet waiting for transmission over a link ℓ is the combined duration of all periods of time during which the packet has been waiting and has been eligible for transmission over ℓ.

Definition 4: The scheduling algorithm $\mathcal{S}_\ell$ of a link ℓ is defined to be *livelock-free* if the eligibility age of no packet waiting for transmission over ℓ can grow indefinitely.

Notice that freedom of livelock, as defined in the above, is not satisfied by all common scheduling algorithms. For example, in a strict priority queue, a packet of low priority could indefinitely wait if higher priority traffic continue to arrive at a sufficient rate.

Theorem 1: Consider a packet network using the selective backpressure protocol **P**. Assume that no packet in the network travels more than D hops. Furthermore, assume that the propagation delays of all links are zero, the network routing is static and packet forwarding in the network is destination-based.[3] In the absence of transmission or processing errors, the following properties hold.

1) Packet transmission in the network is loss free.
2) The order of packets belonging to the same session is maintained as they pass through the network provided that their order would be maintained by the scheduling algorithm $\mathcal{S}_\ell$ of each traversed link ℓ when operating in the absence of protocol **P**.
3) The network is free of deadlock.
4) The network is free of livelock provided that the scheduling algorithms $\mathcal{S}_\ell$ of each network link is livelock-free.

To prove this theorem, we first present three lemmas.

Lemma 1: The level λ_p of a packet p buffered at a given node n always satisfies

$$\lambda_p \leq D \tag{10}$$

with equality only if n is the destination node for p.

Proof: Let $j_n^d(t)$ denote the level that is associated, at node n and time t, with packets going to the destination d. Remember that if there is no entry in the level table of node n for destination d at time t, by definition we have $j_n^d(t) = 0$. Consider an arbitrary destination d, an arbitrary node n_0, and a packet p_0 with the destination d which is waiting at n_0 at some time t_0. Let (t_0', t_0) be the maximal period of time, ending at t_0, during which $j_{n_0}^d(t)$ is both defined and constant. It turns out that at time t_0', the value of $j_{n_0}^d(t)$ must have been either increased to, or set to $j_{n_0}^d(t_0)$, as the result of the arrival of some packet p_1 with the destination d from an upstream node n_1. In either case,

[3]See Section IV for the necessary modifications in protocol **P** to accommodate a network with nonzero propagation delays, adaptive routing, and packet forwarding based on means other than the destination address.

we conclude that the forwarding of p_1 from n_1 to n_0 must be in response to a transmit feedback $f_\ell = j^d_{n_0}(t_0) - 1$ sent from n_0 to n_1. It follows that the level associated with p_1 at node n_1 prior to transmission must have been $j^d_{n_0}(t_0) - 1$ or larger. Therefore, there exists some time $t_1 < t'_0$ for which $j^d_{n_1}(t_1)$ is defined and satisfies

$$j^d_{n_1}(t_1) \geq j^d_{n_0}(t_0) - 1 \, . \tag{11}$$

Applying a similar argument to node n_1, we may conclude that

$$j^d_{n_2}(t_2) \geq j^d_{n_1}(t_1) - 1 \geq j^d_{n_0}(t_0) - 2 \tag{12}$$

for some time $t_2 < t_1$, and for some upstream node n_2 with respect to the destination d. Assume that this process of tracing back the evolution of levels associated with packets destined for d is continued for h steps. At step h of the process, we can write

$$j^d_{n_h}(t_h) \geq j^d_{n_{h-1}}(t_{h-1}) - 1 \geq j^d_{n_0}(t_0) - h \tag{13}$$

for some upstream node n_h and some time t_h. Since network routing is assumed to be static, the node sequence $n_h, n_{h-1}, \ldots, n_0$, must be part of the fixed routing path from n_h to d. Therefore, since no route in the network can be longer than D hops, the above process of backtracing must eventually terminate at some node s for which no upstream node with respect to d exists. At such a node s, all packets destined for d are assigned with level 0 since all of them enter the network at s. Without loss of generality, let $s = n_h$, which implies that

$$h \leq D \tag{14}$$

and

$$j^d_{n_h}(t_h) = 0. \tag{15}$$

Finally, by combining (13)–(15), we obtain

$$j^d_{n_0}(t_0) \leq h \leq D \, . \tag{16}$$

In particular, we notice that, if $j^d_{n_0}(t_0) = D$, then $h = D$, which means that node n itself must be the destination d (for otherwise a route with more than D hops exists). The lemma follows since (16) applies for all choices of d, n_0, and p_0. □

Lemma 2: At the receiving queue associated with any link ℓ, parameters m_i always satisfy

$$m_i = m_{i-1} + b_i - n_i, \quad i = 2, 3, \ldots, D \tag{17}$$

and

$$m_i \geq 0, \quad i = 1, 2, \ldots, D. \tag{18}$$

Proof: Equation (17) follows directly from the definition of m_i in (8). To verify (18), first notice that m_i's are initially positive due to the requirements in (9) and (3). Moreover, according to the buffer management rules of protocol **P**, at the receiving queue of any link ℓ, m_i, $i = 1, 2, \ldots, D$, may be decreased by γ only when a packet of length γ and level $j \leq i$ arrives. But, according to the transmission rule of **P**, such a packet will not arrive unless a transmit feedback $f_\ell \leq j - 1 < i$, has been sent, which indicates that currently $m_i \geq \gamma_{\max} \geq \gamma$. Therefore, m_i remains positive even after it is reduced by γ. □

Lemma 3: Consider the receiving queue associated with a network link ℓ and an arbitrary level k, $1 \leq k \leq D$. Assume that each packet in the buffer that has a level k or higher, will leave the buffer within some finite time. It follows that for any arbitrary time t_0, there is a finite time $t_1 > t_0$, at which $f_\ell < k$.

Proof: We use induction on k to prove the lemma. First, we prove the lemma for $k = D$, by contradiction. Let t_0 be an arbitrary time. Assume that each packet of level D that is in the queue at t_0, leaves the queue by some finite time $t_1 > t_0$, but f_ℓ is always equal to D, after t_0. It follows from the transmission rule of protocol **P** and from Lemma 1 that, after t_0, no new packet may arrive over link ℓ. Therefore, at t_1, there are no packets of level D in the buffer, implying that $n_D = 0$. According to (3), (17), and (18), at t_1 we have

$$m_D = m_{D-1} + b_D - n_D \geq \gamma_{\max}. \tag{19}$$

We conclude from the feedback rule that at time t_1 $f_\ell < D$, contradicting the assumption that $f_\ell = D$, after t_0. This establishes the lemma for $k = D$.

Next, we show that, if the lemma holds for $k + 1$, it must also be true for k, $k = 1, 2, \ldots, D - 1$. We again prove this claim by contradiction. Let t_0 be an arbitrary time. Assume that:

1) the lemma holds for $k + 1$;
2) all packets of level k or higher that are in the buffer at time t_0 will leave the buffer before some finite time $t_2 > t_0$;
3) after t_0, we always have $f_\ell \geq k$.

In view of assumptions 1) and 2), we can apply the lemma to level $k + 1$ and time t_2 to conclude that there is some time t_3, $t_3 > t_2$, at which $f_\ell < k + 1$. It follows from the feedback rule that at time t_3

$$m_j \geq \gamma_{\max}, \quad j \geq k + 1. \tag{20}$$

Also, from assumption 3) and the transmission rule of protocol **P**, it follows that, after t_0, no new packet arriving over link ℓ may be designated with level k. Therefore, in view of assumption 2), at time $t_3 \geq t_2$, there will be no packets of level k in the buffer and $n_k = 0$. Using (3), (17), and (18), we conclude that at t_3

$$m_k = m_{k-1} + b_k - n_k \geq \gamma_{\max}. \tag{21}$$

It follows from (20), (21), and the feedback rule that at time t_3 $f_\ell < k$, which contradicts assumption 3). This completes proof of the lemma. □

Proof of Theorem 1: **Part 1**: No packet will be transmitted toward a node n unless a transmit feedback $f_\ell \leq D - 1$ has been received from it. Therefore, for the corresponding input buffer we have $m_D \geq \gamma_{\max}$. Since m_D is the size of the vacant buffer space, enough buffer is available to store the packet.

Part 2: Consider a node n, and the set of packets stored at n at a given time t that are to be sent over the same outgoing link ℓ and toward the same destination d. According to the protocol, all of these packets have the same level. Therefore, at any given time t, either they are all eligible or all ineligible for transmission over link ℓ. It follows that, for any given scheduling algorithm S_ℓ, the *relative* order of transmission at link ℓ of packets going to the same destination will be identical in the presence or absence of protocol **P**. Since by assumption, in the absence of protocol **P**, the order of packets of each given session is preserved when passing through a a link ℓ, the same thing would be true in the presence of protocol **P**.

Parts 3 and 4: We first claim that, for each network node n and each level k, $k = 1, 2, \ldots, D$, any packet of level k waiting at n will leave n in finite time. To prove this claim, we use induction on k. For $k = D$, our claim follows from Lemma 1, since a packet of level D must be at its destination node and will be delivered to its destination in finite time.

Next, we show that, if our claim is true for all network nodes and packets of level $k+1$, the same will be true for packets of level k. Consider a packet p of level k waiting at some node n, at an arbitrary time t_0. If n is the destination node, p will leave n in finite time. So, let p be waiting for transmission over a link ℓ to the downstream node n'. By assumption, all packets of level $k+1$ or higher that are waiting at any network node, will leave that node in finite time. Therefore, we can apply Lemma 3 to the receiving queue of link ℓ at node n' and time t_0 to conclude that the Feedback parameter f_ℓ will be less than $k+1$, at some finite time $t_1 > t_0$. Similarly, by applying Lemma 3 to moments of time after t_1, we can conclude that f_ℓ will repeatedly be less than $k+1$. It follows that, while p is waiting at node n, it will be eligible for transmission over ℓ, repeatedly (if not continuously). For part 3 of the theorem, we can assume that there are no new packet arrivals to the network after some point. Therefore, the number of packets that must be sent over ℓ is finite, implying that packet p will leave node n in finite time. For part 4 of the theorem, since the scheduling algorithm $\mathcal{S}_\ell$ is livelock-free, p will leave n in finite time, for otherwise its eligibility age will grow indefinitely. This completes the induction proof for our claim.

Finally, we notice that since each packet may travel a bounded number of hops in the network, and since the waiting time of each packet at each node is bounded, each packet will leave the network in finite time. □

IV. Variations of Protocol **P**

In Section III, we developed and studied protocol **P** while limiting our attention to networks with static routing, destination-based packet forwarding, and zero propagation delays. It is now time to relax these assumptions and see how the basic form of the protocol must be modified to work in more general networking scenarios—a task that we will undertake in Sections IV-A–C.

A. Compatibility With Adaptive Routing

In this section, we address the issue of adaptive routing and see how the protocol **P** must be modified to preserve its properties in the presence of routing updates in the network.

As the first step toward establishing the properties of protocol **P**, we proved in Lemma 1 that when **P** is used in a network with static routing, the level assigned at any node n to a destination d is always less than D, unless $d = n$, where D is the maximum number of hops traveled by any packet in the network. We now use a simple example to show that this property does not hold when the routing is not static and changes with time. Consider the network of Fig. 7 and the traffic originating at node a and destined for node d. Assume that, initially, this traffic is sent along the path $abcd$, and let the routing path be later changed to $acbd$. Since both of these routes are three hops long, for this small network and in the absence of longer routes for other traffic, we can set $D = 3$. We further assume that,

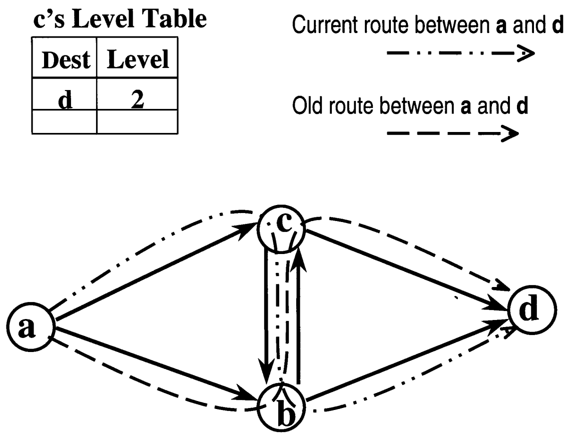


Fig. 7. Interaction of protocol **P** with adaptive routing. Node c's current level table reflects residual information from an old route between a and d.

while the first route $abcd$ is still in use, the level associated with the destination d at node c reaches its maximum attainable value of 2 and remains at this value until the routing changes. It is clear that, after the routing update, the new packets arriving at c for the destination d will continue to be assigned with level 2, even though they now only travel one hop from the origin, in order to reach c. Let p be one such packet and assume that node c forwards it to b at a time when the feedback f_ℓ received from node b is equal to 2. Packet p, upon arrival to b, will be assigned with the level $D = 3$, even though it has not reached its destnation. The above example illustrates that, when adaptive routing is employed, Lemma 1 could be violated. This potential problem cannot be completely rectified by simply increasing the cap D on the packet levels that the protocol views as legitimate. In fact, in this example, it is easy to see that the packet levels can grow indefintely if the routing is repeatedly alternated between $abcd$ and $acbd$.

The reason for this undesired increase in the packet levels is the old information retained in the level tables and the protocol **P**'s principle of keeping the level of all buffered packets, with a common destination, identical. This principle was adopted in order to simplify implementation and to avoid misordering of packets belonging to the same session.[4] As we will now see, packet ordering can be retained by adopting a somewhat different approach that is also compatible with adaptive routing. Assume that a packet p with the destination d arrives to a node n at a time when the feedback for the corresponding link is f_ℓ and there are already some other packet(s) at n, with the destination d and level λ. In accordance with the protocol **P**, if $\lambda > 1 + f_\ell$, the new packet should be assigned with the level λ. Note, however, that it is also possible to assign the level $1 + f_\ell$ to p without disturbing the order of packet transmissions out of node n, since $1 + f_\ell$ is less than the level of the older packets in the buffer. On the other hand, if $\lambda < 1 + f_\ell$, the new packet should be assigned with the level $1 + f_\ell$, and in order to prevent packet misordering, all the previously arrived packets with a lower level (who belong

[4]It is of course possible to regularly refresh the level tables by deleting (i.e., reseting to zero) each entry, once all packets going to the corresponding destination have departed. However, such a remedy does not always prevent the problem of undesired level growth since the routing updates at a node need not coincide with the periods of time when the buffer is empty.

to the same destination), should be lifted to level $1 + f_\ell$. Notice that, with the modification just suggested, protocol **P** continues to ensure proper packet ordering while avoiding the undesirable increase in the level of newly arriving packets at a node, caused by the high level of some previous packet.

With the above change in the protocol, currently buffered packets at a given node, which share a common destination, could have different levels. This constitutes a departure from our earlier view of associating a single level with a destination. In summary, in order to accommodate routing updates in the network, we change the level assignment rules of protocol **P** as follows.

Level Assignment Rules:

At each node n, when a packet p with the destination d arrives from another network node over some link ℓ:

1) assign p with the level

$$\lambda_p = 1 + f_\ell \tag{22}$$

where f_ℓ is the value of the most recent transmit feedback sent over the reverse link ℓ';

2) lift the level of all packets p' that have the same destination d and which are currently buffered at n as

$$\lambda_{p'} \longleftarrow \max(\lambda_p,\ \lambda_{p'}). \tag{23}$$

We now illustrate these rules with an example. Consider the consecutive arrival of four packets, with the common destination d, to node n. Assume that the feedback f_ℓ at the arrival time of these packets is 4, 1, 3, and 0, respectively. Let the last packet arrive when the previous three packets are still queued at n. Each packet, upon arrival, will be assigned with the level $1 + f_\ell$, i.e., levels 5, 2, 4, and 1, respectively. When the second packet arrives, the level of the first packet remains unchanged. Therefore, the first two packets in the queue will have the levels 5 and 2, respectively. When the third packet arrives, the level of the second packet must be lifted to 4 (to match the level of the newly arriving packet and prevent potential misordering), but the level of the first packet remains unchanged since it is already higher than 4. This brings the level of the first three packets in the buffer to 5, 4, and 4, respectively. Finally, when the fourth packet arrives, the levels of all the previous packets remain unchanged. So, we end up with levels 5, 4, 4, and 1, respectively, for the four packets (of destination d) in the buffer.

As the above example shows, the levels associated with the packets belonging to the same destination, always form an ordered list: older packets in the buffer have levels greater than or equal to levels of newer packets. Proper packet sequence is therefore maintained. Also, old packet level information is automatically deleted as packets are transmitted.

B. *Packet Forwarding Considerations*

So far, we have been addressing networks with *destination-based packet forwarding*, i.e., networks in which, at every node, a packet's *destination address* determines the next hop to which it is forwarded. In such networks, in order to maintain packet ordering, it sufficed to insist that all packets going to the same destination should have the same level, thereby the notion of *Destination level*. Alternatively, when in Section IV-A we had to relax the notion of Destination levels, packet ordering could be maintained by subjecting packets with a common destination to the level assignment rule of (23). In both of these level assignment approaches, packets with a common destination had to be treated in a common way, because it was the destination address that determined the next hop to which a packet would be forwarded.

It is straightforward to adapt our protocol to networks where next-hop selection for packet forwarding is performed in a different way. For example, in ATM, where the next hop is determined based on a packet's virtual circuit (VC) number, we simply need to replace the term "destination" in the level assignment rules with the term "virtual circuit," in order to maintain packet ordering. Also, since virtual circuits in ATM are unaffected by routing updates (which are only applied to new VC's), we can follow the approach of Section III-A and assign a common level to all packets of the same VC. The net result is that, for ATM, the notion of *destination level* is replaced with the notion of *VC level*, and a node's level table records the level associated with each active VC going through the node. Similarly, in a network based on MPLS technology where packet forwarding is accomplished using the (top of the stack) label included in each packet, a level must be associated with each *label*, rather than each destination, so that all packets bearing the same label would have the same level, thus maintaining their sequence.

Finally, we conclude this topic by mentioning that if in any of the versions there is no need or desire to maintain packet sequence, then each packet's level can be assigned individually and independent of all other packets, be it packets of the same destination, the same VC, etc. In this case, none of the above considerations with respect to the type of routing or packet forwarding would be necessary. Once a packet arrives to a node n, all that is needed is to assign it with the level $1 + f_\ell$ (or level 0 if the packet is just entering the network) and keep this level unchanged until the packet leaves that node. Accordingly, the notions of destination level or VC level would disappear.

C. *Incorporation of Nonzero Propagation Delays*

Up to this point, we have ignored the effects of propagation delays. To incorporate the effects of a nonzero round-trip delay T in the original protocol **P** (i.e., without the modifications of Sections IV-A and B), we simply need to redefine (re-interpret) the n_i occupancy parameters to include (worst-case) potential packets of level i that *might be* received at R_ℓ during the next T time units. In other words, n_i includes the combined lengths of level i packets in the receiving queue of link ℓ (as before), *plus* those it might receive during the next T time units.

This is accomplished in the following manner. Whenever the transmit feedback level $f_\ell(t) = j$, we simply increase $n_{1+j}(t)$ at rate r (where r denotes the link rate from X_ℓ to R_ℓ). This forces the level $(1+j)$ occupancy parameter n_{1+j} to grow as if packets of level j (or greater) are actually transmitted (by X_ℓ), without interruption, in response to the feedback signal. T units later, if a packet is *not* received at R_ℓ, then $n_{1+j}(t)$ is simply reduced at rate r (during link ℓ's idle periods).

With this new interpretation of the n_i (and the resulting m_i buffer management parameters), the Transmit Eligibility and Transmit Feedback Rules are unchanged, and the (modified)

Level Assignment Rule at time t_0 is based on the transmit feedback value that was sent at time $t_0 - T$ (which is the value X_ℓ had received when it started to transmit the packet).

Level Assignment Rule: When a packet p with destination d starts to arrive at time t from another network node over some link ℓ, the level associated with d is updated as

$$\lambda^d(t) \longleftarrow \max\left(\lambda^d(t),\ 1 + f_\ell(t-T)\right). \tag{24}$$

Note that R_ℓ needs to maintain history of the transmit feedback signals $f_\ell(t)$ for T time units (similar to the "book-keeping" algorithms described in [11] and [12]). Actually, it just needs to maintain a list of integer feedback values and the corresponding time instants they changed. Also, if an arriving packet is assigned a level equal to the existing (larger) $\lambda^d(t)$ rather than $1 + f_\ell(t-T)$, then the two corresponding occupancy parameters need to be appropriately increased and decreased, respectively.

What happens if R_ℓ does not know the exact value of T? Then either (i) unused "guard times" could be inserted into the link protocol to compensate for uncertainty in the value of T, or (ii) X_ℓ could insert "acknowledgment signals" between its transmissions (or during idle periods) on the (forward) link ℓ to acknowledge transition points in its received transmit feedback values. The right solution probably depends on how large the uncertainty in T is compared with the packet transmission time.

Finally, if the b_i buffer budgets are too small (for large values of T), then the stop-and-go nature of protocol **P** might (temporarily) interrupt the transmission of packets even in the absence of congestion and buffer overload. Consequently, we suggest keeping $b_1 \geq rT + \gamma_{\max}$ (i.e., large enough to buffer at least one round-trip delay worth of packets). This ensures that, in the absence of congestion and buffer overload, transmission of packets may continue without interruption.

V. Alternatives and Extensions

In this section, we present a few possible extensions of the protocol. First, while this new technique addresses the problems of short-term congestion overflows and deadlocks (in networks such as gigabit ethernet), it does not address the important issue of fairness in providing service to different users. One way of resolving this shortcoming is to couple this technique with end-to-end congestion control schemes that handle congestion problems on a quasi-static basis while providing the desired fairness and/or priorities in the amount of services given to different users in the long run (e.g., [13], [14]).

Second, it is sometimes advantageous to incorporate some packet dropping to address other network issues, such as aging of packets due to errors, and blocking caused when certain network links are "overloaded." Specifically, although protocol **P** avoids packet loss in the prevention of deadlocks and livelocks, it may need to be coupled with the packet dropping that is used to address other network issues. For instance, packet dropping is necessary to remove packets that suffer errors and are no longer deliverable to their destinations. Perhaps more interesting, though, is the need to drop some packets when certain network links become "overloaded." Although the selective backpressure technique prevents deadlocks and livelocks, the (sustained) overload condition might spread to other parts of the network and reduce its throughput. To address this problem, some of the "offending" packets should be selectively dropped from the congested links. Suppose, for example, in Fig. 4 that "too much" traffic is arriving destined for output port 3, with many of those packets arriving at input port 1. Then, the increased occupancy of receiving queue 1 (due to the increased time it takes to drain sending queue 3) will cause input port 1 to be too restrictive in the packets that it allows its upstream neighbor to transmit. If receiving queue 1 is filled with "too many" packets destined for output port 3 (i.e., the queue is filled in an "unfair" manner with respect to the various output links), then it may be helpful to selectively drop some of the "offending" packets to resolve the memory-sharing problem. Performance of the protocol **P** (which reduces end-to-end delay when network resources are used in a "fair" fashion) coupled with selective dropping of packets that take "too much" of a node's memory is a topic for future research.

Third, if it is desired to internetwork the proposed lossless network with a network (e.g., TCP) that depends on losses to rate control its sources, then it is simple to design a "gateway" between the two networks. For instance, an edge device near a TCP source or near the WAN, could be used to convert the lossless LAN's end-to-end technique to the TCP "loss technique" (i.e., by dropping packets) to implicitly rate control the TCP sources.

Finally, the protocol can be modified such that some classes of service (CoS) will be allowed to "ignore" the transmit feedback congestion control signals that gradually restrict the set of packets eligible for transmission. This is possible if, perhaps, there are dedicated buffers set aside for them. Alternatively, all nodes might be programmed to always treat particular CoS packets to be of no less than a certain minimum level.

VI. Conclusion

In this paper, we described a simple, lossless method of preventing deadlocks and livelocks in backpressured networks. In contrast with prior approaches, our proposed technique prevents deadlocks and livelocks without introducing any packet losses, without corrupting packet sequence, and without requiring any changes to packet headers. This makes it possible to apply the technique to full-duplex connections in gigabit ethernet (IEEE 802.3z) [10]. It represents a *new networking paradigm* in which internal network losses are avoided (thereby simplifying the design of other network protocols) and internal network delays are bounded.

References

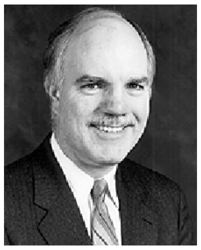
[1] W. Noureddine and F. Tobagi, "Selective back-pressure in switched ethernet LAN's," in *Proc. IEEE GLOBECOM*, Dec. 1999, pp. 1256–1263.

[2] P. M. Merlin and P. J. Schweitzer, "Deadlock avoidance in store-and-forward networks—I: Store-and-forward deadlock," *IEEE Trans. Commun.*, vol. COM-28, pp. 345–354, Mar. 1980.

[3] K. D. Gunther, "Prevention of deadlocks in packet-switched data transport systems," *IEEE Trans. Commun.*, vol. COM-29, pp. 512–524, Apr. 1981.

[4] I. S. Gopal, "Prevention of store-and-forward deadlock in computer networks," *IEEE Trans. Commun.*, vol. COM-33, pp. 1258–1264, Dec. 1985.

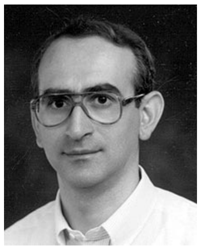
[5] I. Cidon, J. M. Jaffe, and M. Sidi, "Distributed store-and-forward deadlock detection and resolution algorithms," *IEEE Trans. Commun.*, vol. COM-35, pp. 1139–1145, Nov. 1987.
[6] M. D. Schroeder *et al.*, "Autonet: A high-speed, self-configuring local area network using point-to-point links," *IEEE J. Select. Areas Commun.*, vol. 9, pp. 1318–1335, Oct. 1991.
[7] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. C-36, pp. 547–553, May 1987.
[8] E. Leonardi *et al.*, "Congestion control in asynchronous, high-speed wormhole routing networks," *IEEE Commun. Mag.*, pp. 58–69, Nov. 1996.
[9] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[10] M. Karol, S. J. Golestani, and D. Lee, "Prevention of deadlocks and livelocks in lossless, backpressured packet networks," in *Proc. INFOCOM*, Mar. 2000, pp. 1333–1342.
[11] I. Iliadis, "A new feedback congestion control policy for long propagation delays," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1284–1295, Sept. 1995.
[12] I. Iliadis and R. Marie, "Evaluation of a start-stop protocol for best-effort traffic in ATM networks," *Comput. Commun. Rev.*, vol. 21, pp. 1544–1588, 1998.
[13] R. G. Gallager and S. J. Golestani, "Flow control and routing algorithms for data networks," in *Proc. 5th Int. Conf. Computer Communications*, 1980, pp. 779–784.
[14] S. J. Golestani and S. Bhattacharyya, "A class of end-to-end congestion control algorithms for the Internet," in *Proc. 6th Int. Conf. Network Protocols*, Oct. 1998, pp. 137–150.



Mark Karol (S'79–M'85–SM'91–F'93) received the B.S. degree in mathematics and the B.S.E.E. degree from Case Western Reserve University, Cleveland, OH, in 1981 and the M.S.E., M.A., and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, in 1982, 1984, and 1986, respectively.

From 1985 to 2000, he was a member of the Research Communications Sciences Division, Bell Laboratories, Holmdel, NJ. Since September 2000, he has been a member of Avaya Labs–Research, Lincroft, NJ, where he is currently a Distinguished Member of Technical Staff, working on the design and analysis of next-generation switching and networking products.

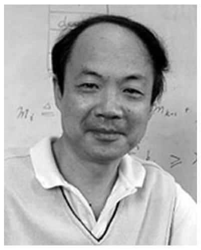
In 1993, Dr. Karol became a Fellow of the IEEE for "contributions to the fundamental theory, design, and analysis of high-performance packet switches and multiuser lightwave communication networks." He has served as an associate editor for the *Journal of Lightwave Technology*, General Chair of the 2002 IEEE International Conference on Communications (ICC2002), General Chair of the IEEE INFOCOM'94 Conference, and Director of Magazines for the IEEE Communications Society.



S. Jamaloddin Golestani (M'85–SM'95–F'00) was born in Iran in 1955. He received the B.S. degree from Sharif University of Technology, Tehran, Iran, in 1973 and the M.S., E.E., and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge, in 1976, 1976, and 1979, respectively, all in electrical engineering.

He spent the next eight years at Isfahan University of Technology, Isfahan, Iran, where he taught and did research in communication theory, communication networks, and radar systems. From 1988 to 1995, he worked at Bell Communications Research, Morristown, NJ, where he was a member of the Information Networking Research Laboratory. Since 1995 he has been working at Bell Laboratories, Lucent Technologies, Murray Hill, NJ, as a Member of the Networking Research Laboratory. His research interests include distributed network algorithms, scheduling algorithms, congestion control, routing and wireless networks. Much of his past research was devoted to the formulation of congestion control problem and provision of fairness and guaranteed services in packet networks.

Dr. Golestani has previously served on the Editorial Board of the IEEE/ACM TRANSACTIONS ON NETWORKING.



David Lee (M'89–SM'91–F'97) received the M.S. and Ph.D. degrees in computer science from Columbia University, New York, NY, in 1985.

He is currently a Vice President of Bell Labs Research, Murray Hill, NJ. He is a Counselor of the National Natural Science Foundation of China. His current research interests are secure and reliable mobile wireless network systems and protocol system integration, interoperability and reliability.

Dr. Lee received the IFIP International Conference PSTV-FORTE Best Paper Award in 2000. He is an Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, a Senior Editor of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, an Associate Editor of the *Journal of Complexity*, and serves on the Editorial Board of *Electronica Sinica* and the *Journal of Advanced Networks*. He was Program Co-chair of the IEEE INFOCOM 2002, ICNP 1994 and 2000, General Co-chair of ICNP 1996, and Steering Committee Co-chair of ICNP.