Prevention of Deadlocks and Livelocks in Lossless, Backpressured Packet Networks

Mark Karol, S. Jamaloddin Golestani, David Lee
Lucent Technologies
Email: mk,golestani,leedavid@lucent.com

Abstract— **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. One way to accomplish this goal is to have the congested nodes send backpressure feedback to neighboring nodes, informing them of unavailability of buffering capacity and in effect stopping them from forwarding 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. In addition to presenting the new congestion control protocol in a general context, we describe an important application of the technique to gigabit ethernet (IEEE 802.3z).**

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 applications transported by the network may not meet their quality-of-service (QoS) requirements. Worse yet, congestion can potentially lead to deadlocks and livelocks in the network. A *deadlock* is a condition under which the throughput of the network, or part of the network, goes to zero due to congestion (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. In a *livelock* situation, the network is not stopped, but 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.

Many different types of deadlocks have been identified and studied, along with methods of preventing them (see, e.g., [1], [2], [3], [4], [5], [6]). Figure 1 shows a simple example in which the cycle of nodes - X, Y, and Z - are in a deadlocked state. Note that X, Y, and Z can be any three adjacent nodes that form a "cycle" in a larger network of nodes.

Proper congestion control is especially an important topic in emerging local area networks (LANs): large LANs with a heterogeneous mix of link speeds (ranging, for example, from 10 Mbps up to 1 Gbps) and the need to support the quality-of-service (QoS) requirements of multimedia applications from hundreds or even thousands of users. Typically, when congestion occurs in packet networks, packets are simply dropped from the congested buffers. Dropped packets are later retransmitted by end-to-end protocols.

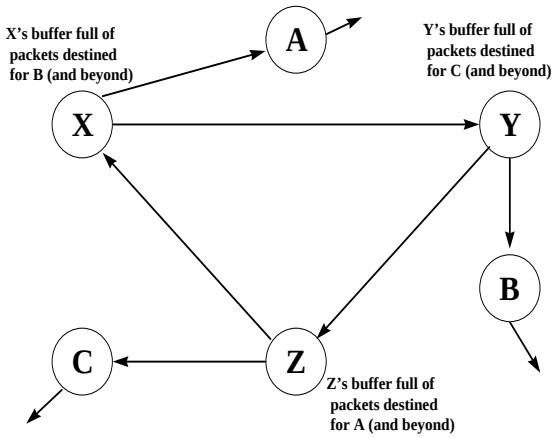

Fig. 1. Deadlock Example

In the LAN context, recent simulation results show that hop-by-hop backpressure can be better than TCP for dealing with short-lived congestion [7]. TCP, the dominant transport protocol in the Internet, uses packet drops as an indication of congestion and requires sufficient levels of loss in order to be an effective control. In a sense, TCP keeps increasing the load in order to increase the loss so that the necessary feedback signals are sent. Hop-by-hop backpressure helps reduce the number of packets dropped during periods of transient congestion, and avoids wasting the network resources so far already consumed up to this point. This is particularly important when we consider packets that might arrive at a LAN after traversing a WAN (see Fig. 2). The penalty is that some links are "turned off" for short periods of time, perhaps negatively impacting other flows (i.e., those not involved in the "congestion") as the backpressure propagates through the network. The advantage is that the distributed memory resources in a network can be used to buffer excess traffic generated by bursty sources. This helps bypass the costly TCP flow control mechanism, which may have a negative performance impact in the LAN context [7].

In this paper, we describe a simple, *lossless* method of preventing deadlocks and livelocks in packet networks that use

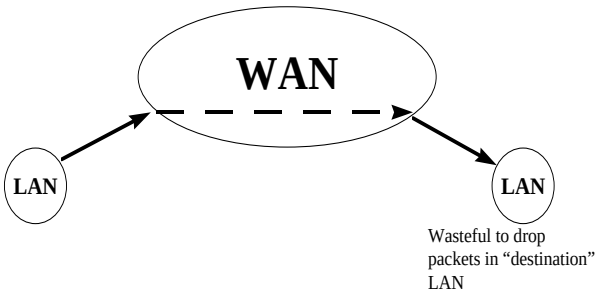

Fig. 2. Avoid Dropping Packets After Traversing WAN

"stop-start" (off-on) types of algorithms to control congestion. Stop-Start congestion control is a MAC layer technique that relies on hop-by-hop backpressure [8]. Researchers have known for many years that deadlocks (and livelocks) can occur when these types of congestion control are used in networks. Stop signals can, in theory, loop back on themselves, creating a deadlocked "cycle" as the backpressure propagates through the network (see, e.g., Fig. 1). Although a stop-start congestion control mechanism can, in theory, eliminate packet loss in networks, this capability is obviously gone if packets need to be dropped to prevent deadlocks or to recover from a deadlock condition. In contrast with prior approaches, our proposed technique prevents deadlocks and livelocks in backpressured networks *without introducing any packet losses*. In addition, our protocol maintains packet sequence and requires no changes to packet headers.

In addition to presenting the new congestion control protocol in a general context, we describe an important application of the technique to gigabit ethernet (IEEE 802.3z). Full-duplex connections in a gigabit ethernet network can take advantage of the explicit "stop-start" congestion control provided by IEEE 802.3x, in which a PAUSE frame stops the flow of data frames for specified periods of time (indicated by a parameter in the PAUSE frame). A PAUSE frame that contains a parameter of zero time allows the flow of data frames to restart immediately. In this paper, although we focus our examples on gigabit ethernet, the proposed congestion control techniques are more general.

Before reviewing some existing deadlock prevention techniques for backpressured networks, we note that proper strategies for dealing with deadlocks increase in importance as data transmission rates increase to gigabit-per-second (and higher) rates. For a given network load level, the number of (potential) deadlocks per hour (that have to be prevented, avoided, or recovered from) increases in proportion to the transmission rate. In addition, (potential) deadlocks will occur more frequently as the network loading increases. So, for instance, as gigabit ethernet links are extended to cover greater distances in metropolitan-area networks (MANs) and WANs, heavier loading of long-distance links (to make efficient use of the links) will cause (potential) deadlocks to occur more frequently. Currently, proprietary hardware and high-quality fiber-optic lines can extend the distances between switches to greater than 70 km, permitting gigabit ethernet implementations across MANs and even WANs [9], [10].

In Section II, we review some existing deadlock prevention techniques that can be used with stop-start algorithms. In Section III, we present a new, simple, lossless method of preventing deadlocks and livelocks in packet networks. We first proposed the technique for gigabit ethernet in [11]; here we include several generalizations of the protocol and proofs that it is deadlock-free and livelock-free. In Section IV, we discuss some remaining important implementation issues, and in Section V we discuss some other alternatives and extensions. Finally, Section VI includes some conclusions of this work.

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 to avoid a deadlock or recover from a deadlock condition. The impact on total end-to-end delay, including interactions with higher-layer protocols (such as TCP), needs to be considered. Also, retransmission of packets might even cause the "deadlock condition" to redevelop.

As mentioned in the Introduction, we focus on preventing deadlocks and livelocks in backpressured networks (such as full-duplex gigabit ethernet) without introducing any packet losses. Before describing our protocol, we first mention a few prior methods that have been proposed to prevent deadlocks.

First, 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.

Second, up/down routing on a spanning tree [12] avoids deadlocks by assigning directions to the links such that cycles are avoided. 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 (which can lead to deadlocks) is to split each physical link into a number of virtual channels, each with its own queue and stop-start backpressure protocol [13] [14]. 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. Extra buffers are 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 head-

ers, 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 Fig. 3 for an example of organizing node memory into buffer classes [15]. In Fig. 3, D represents the maximum number of hops in any legitimate network route. Buffer class k can be used only for packets that have traveled k or more links. Packets that travel more than D links are discarded as having traveled further than should be possible. A proof by induction (starting with $k = D$) shows that the buffers of class k cannot fill up permanently, and therefore no deadlock occurs.

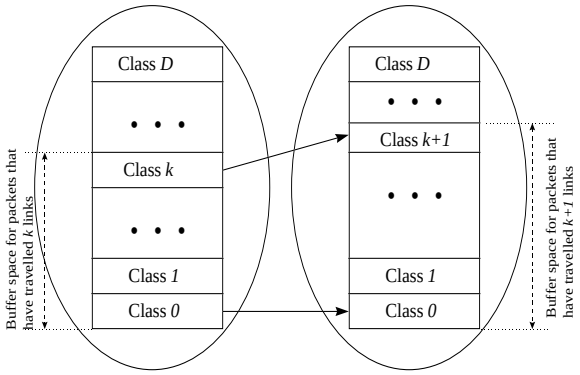

Fig. 3. Buffer Allocation To Avoid Deadlocks

Although these prior schemes that reserve buffer space for various hop-count values are proven to be deadlock-free, little mention is made of specific link scheduling algorithms or network signaling protocols. Many possibilities exist. For instance, packets could simply be dropped (and later retransmitted) if appropriate buffer classes are not available when they reach a node, or signals could be sent to inform an upstream neighbor which classes of packets can be sent.

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. Most 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 ("non-standard") 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. In Fig. 3, for instance, it is possible that a session's packets might be transmitted out-of-sequence to the next node because they were stored in different buffer classes (even though they all arrive with the same hop-count). Specific information would need to be kept regarding the order in which packets were stored in the various buffer classes. In our protocol, all packets with the same *Destination Address* (DA) will be assigned the same *Level* (to be defined later) so that a session's proper packet sequence will be automatically preserved.

A third problem with prior approaches is that although they resolve the deadlock problem, some do not eliminate the possibility of livelock. For example, in Fig. 3, if the scheduling algorithm and signaling protocol are not carefully designed, then in the continual presence of new arrivals with higher hop counts, packets in lower buffer classes might never have a chance to be transmitted. In Section III-B, we discuss some conditions for the scheduling algorithm that guarantee livelocks and deadlocks are both prevented in our protocol.

A fourth problem with prior approaches is that network nodes need some type of "signaling message" to tell upstream neighbors which packets they can transmit (alternatively, they need a way to send negative acknowledgements when packets are dropped). Provision for such a signaling message does not exist in the IEEE 802.3z standard. In Section IV, we present some possible solutions to this problem.

Finally, a fifth 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. A 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'.

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 I, 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 I. The proposed technique makes use of only the *Des-*

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

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 non-discriminatory 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 $\mathcal{S}_\ell$. 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. As the congestion at node R_ℓ subsides, the restriction placed on the transmission of packers over ℓ is gradually lifted by designating more packets as *eligible* using a new Transmit Feedback f_ℓ.

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. 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.

In a real network, it takes some time until a Transmit Feedback f_ℓ sent by R_ℓ reaches X_ℓ. Likewise, the effect of eligibility designations made at X_ℓ, as the result of the Transmit Feedback received from R_ℓ, does not reach R_ℓ until after a delay related to the propagation time of ℓ. In this section, in order to focus on the essence of protocol **P** and not be sidetracked by secondary issues, we assume that the propagation delays of all network links are zero and that the Transmit Feedback generated at a receiving node R_ℓ is instantaneously sent to and detected by the sending node X_ℓ. Later, we will discuss necessary modifications in the protocol to accommodate real network scenarios where these conditions are 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, 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. The selective backpressure protocol **P** is based on the following rules for the assignment and updating of these destination Levels.

Level Assignment Rules:

1. 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.
2. 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.
3. At any point of time, the Level λ_p of each packet p which is buffered at node n is equal to the Level λ^d associated with the corresponding destination d, at that time.
4. When a packet p with destination d arrives from another network node over some link ℓ, the Level associated with d is updated as

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

where f_ℓ is the value of the most recent Transmit Feedback sent over the reverse link ℓ'.
5. 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.
6. 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. Note from rule 2 above that such elimination is equivalent to resetting λ^d to zero. It also serves to automatically "refresh" entries in the Level Table, which is needed since the topology or routing paths may change over time.

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 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 with a Level of at least 1, since the Level associated with its destination will undergo the update in (2).
- Updating according to the above rules will never result in a Level larger than D. As will be shown later, by the time the Level associated with a packet reaches D, the packet must have reached its destination. 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.

To illustrate how Levels are assigned, Fig. 4 shows a network example of various Levels and Transmit Feedback values.[1] 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 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.

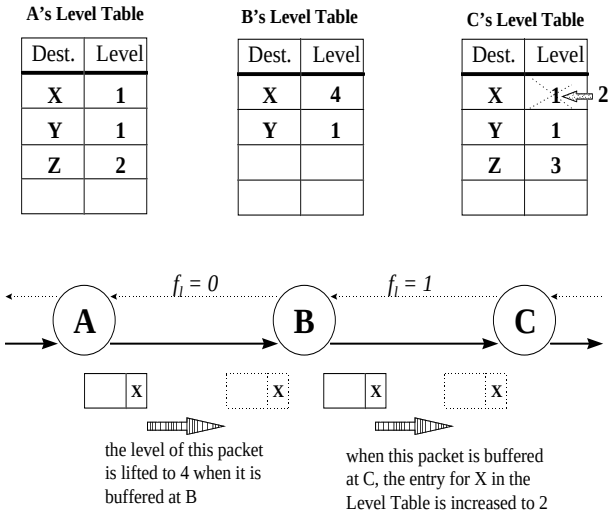

Fig. 4. Example of Levels and Transmit Feedback Values

Now that we have discussed the concepts of packet (or destination) Levels, the Transmit Feedback, and the Transmit Eligibility Rule, we explain in the next few paragraphs how the values for the Transmit Feedback are selected.

For the purpose of our discussion in this section (and to focus on the details of the proofs), we consider an input buffering model for each node of the network.[2] This means that at each network node, a buffer space is associated with each incoming network link. This buffer is exclusively used to store the traffic arriving on that link. Likewise, the traffic entering the network at a node (over some network access link) is stored in a separate buffer assigned for that purpose. Now consider again an arbitrary *network* link ℓ and the receiving node R_ℓ. We refer to the input buffer at R_ℓ that is associated with the incoming link ℓ as the *Receiving Queue* of ℓ. Let the size of this buffer be denoted by b^ℓ, and 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 paragraphs.

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

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

[1] Additional examples are provided in [11].

[2] A more general switch model is described in [11].

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.[3] 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, $i = 1, 2, \cdots, D$, 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

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

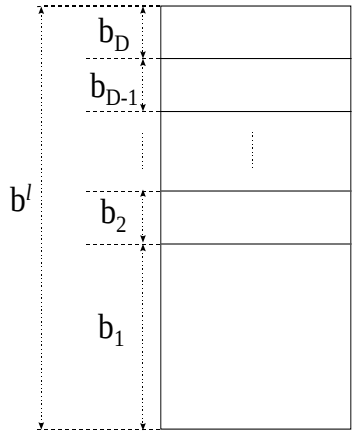

Fig. 5. Budget Allocations of Link ℓ's Receiving Queue

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

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

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,

[3]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."

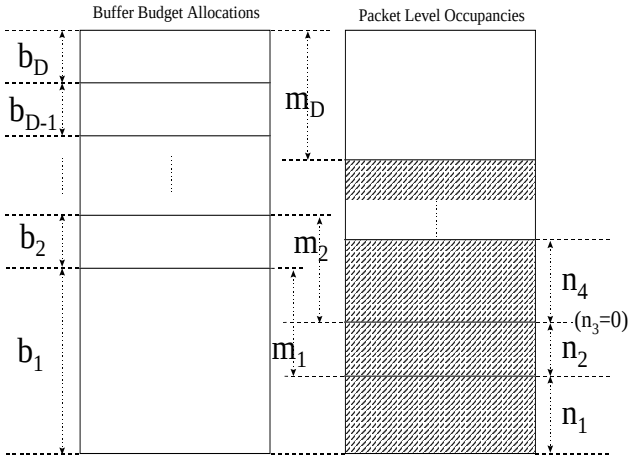

Fig. 6. Buffer Management Parameters - ℓ's Receiving Queue

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 non-negative, 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, 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 $i = 1, 2, \cdots, D$, 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

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

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 Rule: Set link ℓ's Transmit Feedback $f_\ell = j$, where j is the *largest* Level for which $m_j < \gamma_{\max}$. If no such Level exists, set $f_\ell = 0$.

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. Livelock is an extreme example of "unfairness."

Theorem 1: Consider a packet network using the selective backpressure protocol **P**. Assume that no packet in the network travels more than D hops and let the propagation delays of all links be zero. 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, we conclude that the forwarding of p_1 from n_1 to n_0 must be in response to a Transmit Feedback $f_\ell = j_{n_0}^d(t_0) - 1$ 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 $j_{n_0}^d(t_0) - 1$, or larger. Therefore, there exists some time $t_1 < t_0'$ for which $j_{n_1}^d(t_1)$ is defined and satisfies

$$j_{n_1}^d(t_1) \geq j_{n_0}^d(t_0) - 1\ . \tag{11}$$

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

$$j_{n_2}^d(t_2) \geq j_{n_1}^d(t_1) - 1 \geq j_{n_0}^d(t_0) - 2\ , \tag{12}$$

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,

$$j_{n_h}^d(t_h) \geq j_{n_{h-1}}^d(t_{h-1}) - 1 \geq j_{n_0}^d(t_0) - h\ , \tag{13}$$

for some upstream node n_h and some time t_h. Since no route in the network can be longer than D hops, the above process of back-tracing 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 get

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

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:

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

and

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

Proof: (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, $m_i,\ \ i = 1, 2, \cdots, D$, 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 (17), (18), and (3), 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 $k,\ \ k = 1, 2, \cdots, D - 1$. 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 packet 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 (17), (18), and (3), 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 towards 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 towards 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 $k,\ \ k = 1, 2, \cdots, D$, 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 $k+1$, 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. Implementation Issues

In this section, we address some issues regarding the sending of the *Transmit Feedback* signals to the upstream neighbor in gigabit ethernet using the standard PAUSE frames. Included is a discussion of the compatibility with "regular" IEEE 802.3 equipment. Other networks will use other methods to signal congestion status to upstream nodes.

In the protocol **P**, a Transmit Feedback needs to be occasionally sent to an upstream neighbor to tell it which packet Levels it has permission to transmit. It is simple to transmit this information on full-duplex gigabit ethernet connections using standard PAUSE frames. Rather than coding the PAUSE frame's parameter to represent the period of time that the upstream neighbor should not send data frames, the parameter is coded to represent the various Transmit Feedback values.

A slight complication is necessary to allow simple interworking of "regular" 802.3 equipment (i.e., without the enhancements we propose in this paper) with "enhanced" 802.3 equipment. Suppose nodes don't know what types of nodes (i.e., "regular" or "enhanced") they are connected to. When a PAUSE frame is sent from an "enhanced" node to signify a Transmit Feedback value, then the PAUSE frame should be followed immediately by a second PAUSE frame with its parameter set equal to zero (to continue the immediate transmission of data frames).[4] Otherwise, suppose an "enhanced" 802.3 node did not know whether the PAUSE signal it received was coming from a "regular" 802.3 node or from another "enhanced" 802.3 node. It would have to assume that the first PAUSE signal was telling it to stop transmitting for the specified period of time. Upon receiving the second PAUSE signal (with a "zero" value), the "enhanced" node would know that it could immediately resume transmitting data frames that satisfy the constraint expressed by the Transmit Feedback.[5] Finally, if a "regular" node receives the two PAUSE signals back to back from an "enhanced" node, then the "regular" node simply remembers the most recent PAUSE signal (in this case, the one with a parameter value equal to zero) and resumes transmission of its data frames. Provided the second PAUSE signal "immediately" follows the first PAUSE signal, the "regular" node will not even pause its transmission (because a received PAUSE signal does not preempt a node's in-progress transmission of a data frame).

[4] Alternatively, if an "enhanced" node can "learn" that its neighbor is also "enhanced," then it will not be necessary to always send the second PAUSE equals zero frame.

[5] If it had come from a "regular" node, then the "enhanced" node might (unnecessarily) back off some of its transmissions of "lower"-Level packets.

Note that when a network is built with a mixture of both "enhanced" and "regular" IEEE 802.3 equipment, there are no guarantees against the possibility of network deadlocks and livelocks, and packet loss.

V. Alternatives And Extensions

Up to this point, we have ignored the effects of propagation delays. The basic modification needed to incorporate propagation delays into our protocol is to make sure that nodes transmit their feedback signals with enough "lead time" so that the controls "take effect" at the appropriate moments (assuming worst-case conditions). For example, if a node determines that for a given incoming link a particular Transmit Feedback signal needs to take effect at time t_0, then it will need to transmit the signal at time $t_0 - T$, where T is the round-trip propagation time of that link. As the buffer occupancy continues to change, note that a node might need to transmit an updated Transmit Feedback between the time $t_0 - T$ and the time t_0.

Specifically, very little needs to change to incorporate the effects of a round-trip delay T in the protocol **P**. The Transmit Eligibility Rule and the Buffer Management Rules *are unchanged*! Only the Transmit Feedback Rule and the Level Assignment Rules need to be slightly modified.

First, in determining what Transmit Feedback to send at time t_0, the receiving node R_ℓ needs to consider the "worst-case" (maximum) occupancy of its Receiving Queues at time $t_0 + T$ in the future. If the link rate from X_ℓ to R_ℓ is r, then the occupancy of a Receiving Queue may increase by at most rT. Consequently, the modified rule is:

Transmit Feedback Rule: Set link ℓ's Transmit Feedback $f_\ell = j$, where j is the *largest* Level for which $m_j - rT < \gamma_{\max}$. If no such Level exists, set $f_\ell = 0$.

If desired, more complex "book-keeping" algorithms can also be incorporated to make more efficient use of memory and bandwidth resources. Using techniques similar to those proposed in [16] and [17], information about the recent history (from time $t_0 - T$ to time t_0) of feedback signals that were transmitted on a link to an upstream neighbor can be used to compute an upper bound (perhaps, less than rT) on the amount of information that will reach the downstream node between time t_0 and time $t_0 + T$.

Second, the (modified) Level Assignment Rule is based on the *current* buffer occupancy (just as in the $T = 0$ special case considered previously):

Level Assignment Rule: When a packet p with destination d arrives from another network node over some link ℓ, the Level associated with d is updated as

$$\lambda^d \longleftarrow \max\ (\lambda^d,\ 1 + j), \tag{22}$$

where j is *largest* Level for which currently $m_j < \gamma_{\max}$ (if no such Level exists, set $j = 0$).

Note that because we make worst-case assumptions in the (modified) Transmit Feedback Rule, we can assign Levels to packets

at time t_0 without needing to remember what Transmit Feedback value was sent at time $t_0 - T$.[6] This greatly simplifies the implementation.

What happens if the receiving node R_ℓ has an incorrect estimate of the round-trip propagation delay T? If the actual delay is greater than the estimate, then some packet loss may occur and deadlocks are possible. On the other hand, if the actual propagation delay is less than the estimated value used by node R_ℓ, then the only penalty is a (slight) loss in efficiency and perhaps some wasted buffer.

Consequently, in local and metropolitan area network applications where link propagation delays are small, it might be preferable to select some *upper bound* on link propagation delays of the network as the parameter T used on all links - just as there are maximum distances for links due to physical constraints. If the protocol is used in wide area networks, however, it is probably better to individually set the T parameter for each link.

Finally to guarantee no packet loss in the presence of nonzero propagation delays, the buffer size must be increased such that

$$b_1 \geq \gamma_{\max} + rT. \tag{23}$$

Equation (3), constraining b_i, still holds for $i = 2, 3, \cdots, D$.

Another option in dealing with long-propagation environments is to reinterpret some of the Transmit Feedback values (most likely the higher values) as permission to transmit up to a specified maximum number of packets of specified Levels, rather than an unlimited number of packets. Additional Transmit Feedback messages would need to be sent to grant permission to transmit additional packets.

Before briefly presenting a few possible extensions of the protocol, it is important at this point to mention the relationship of our protocol with end-to-end congestion control techniques. While this new technique addresses the problems of short-term congestion overflows and deadlocks, 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., [18]).

Finally, we mention a few more possible extensions of this work. First, 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" [11].

Second, 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.

[6]The Transmit Feedback value j^* at time t_0 is greater than or equal to the j value in the Level Assignment Rule at time $t_0 + T$.

Third, 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. Conclusions

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).

References

[1] E. G. Coffman, M. J. Elphick, and A. Shoshani, "System Deadlocks," *Computing Surveys*, vol. 3, pp. 67-78, June 1971.

[2] R. C. Holt, "Some Deadlock Properties of Computer Systems," *Computing Surveys*, vol. 4, pp. 179-196, Sept. 1972.

[3] 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.

[4] K. D. Gunther, "Prevention of Deadlocks in Packet-Switched Data Transport Systems," *IEEE Trans. Commun.*, vol. COM-29, pp. 512-524, Apr. 1981.

[5] I. S. Gopal, "Prevention of Store-and-Forward Deadlock in Computer Networks," *IEEE Trans. Commun.*, vol. COM-33, pp. 1258-1264, Dec. 1985.

[6] G. J. Holzmann, *Design and Validation of Computer Protocols*. Englewood Cliffs, NJ: Prentice Hall, 1991.

[7] F. A. Tobagi and W. K. Noureddine, "Back-Pressure Mechanisms in Switched LANs Carrying TCP and Multimedia Traffic," submitted to *IEEE GLOBECOM'99 Symposium on High-Speed Networks*, Dec. 1999.

[8] S. Kamolphiwong, A. E. Karbowiak, and H. Mehrpour, "Flow Control in ATM networks: a survey," *Comp. Commun.*, vol. 21, pp. 951-968, 1998.

[9] J. Caruso, "Gigabit Ethernet ventures into the land beyond the LAN," *Network World*, p. 36, May 10, 1999.

[10] N. Margalit, "Intelligent DWDM takes Gigabit Ethernet to the MAN," *Lightwave*, p. 101, June 1999.

[11] M. Karol, D. Lee, and S. J. Golestani, "A Simple Technique that Prevents Packet Loss and Deadlocks in Gigabit Ethernet," *Proc. 1999 International Symposium on Communications (ISCOM'99)*, pp. 26-30, Nov. 1999.

[12] M. D. Schroeder *et al.*, "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," *IEEE Jour. Selected Areas Commun.*, vol. 9, pp. 1318-1335, Oct. 1991.

[13] 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.

[14] E. Leonardi *et al.*, "Congestion Control in Asynchronous, High-Speed Wormhole Routing Networks," *IEEE Commun. Mag.*, pp. 58-69, Nov. 1996.

[15] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice Hall, 1987.

[16] I. Iliadis, "A New Feedback Congestion Control Policy for Long Propagation Delays," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1284-1295, Sept. 1995.

[17] I. Iliadis and R. Marie, "Evaluation of a start-stop protocol for best-effort traffic in ATM networks," *Comp. Commun.*, vol. 21, pp. 1544-1588, 1998.

[18] S. J. Golestani and S. Bhattacharyya, "A Class of End-to-End Congestion Control Algorithms for the Internet," *Proc. 6th International Conf. on Network Protocols*, Oct. 1998.