FLID-DL: Congestion Control for Layered Multicast

John W. Byers, Gavin Horn, Michael Luby, Michael Mitzenmacher, William Shaver

Abstract—

We describe Fair Layered Increase/Decrease with Dynamic Layering (FLID-DL), a new multi-rate congestion control algorithm for layered multicast sessions. FLID-DL generalizes the receiver-driven layered congestion control protocol (RLC) introduced by Vicisano, Rizzo, and Crowcroft, ameliorating the problems associated with large IGMP leave latencies and abrupt rate increases. Like RLC, FLID-DL is a scalable, receiver-driven congestion control mechanism in which receivers add layers at sender-initiated synchronization points and leave layers when they experience congestion. FLID-DL congestion control coexists with TCP flows as well as other FLID-DL sessions and supports general rates on the different multicast layers. We demonstrate via simulations that our congestion control scheme exhibits better fairness properties and provides better throughput than previous methods.

A key contribution that enables FLID-DL and may be useful elsewhere is Dynamic Layering (DL), which mitigates the negative impact of long IGMP leave latencies and eliminates the need for probe intervals present in RLC. We use DL to respond to congestion much faster than IGMP leave operations, which have proven to be a bottleneck in practice for prior work.

Keywords— **IP multicast, layered multicast, congestion control, content delivery, TCP-friendliness, IGMP, scalability.**

I. Introduction

One of the significant remaining hurdles to widespread adoption of IP multicast is the development of suitable congestion control algorithms. Ideally, one would hope for a multicast analogue of TCP congestion control. Such a protocol would be an end-to-end congestion control mechanism that scales to large audience sizes, matches the functional relationship between throughput and packet loss rate at each receiver that TCP achieves, and provides responsiveness to changing network conditions on the order of a round-trip time, like TCP. Challenges include receiver heterogeneity, accurate modeling of TCP performance, and providing compatibility with other transport-level services such as reliability. In this paper, we provide a new multicast congestion control scheme that makes substantial strides toward a deployable solution.

J. Byers is with the Dept. of Computer Science at Boston University and Digital Fountain, Inc., Fremont, CA and is supported by National Science Foundation grant ANIR-9986397 and NSF CAREER award ANIR-0093296. E-mail: byers@cs.bu.edu. G. Horn is with Pulsent Corporation and performed this research while with Digital Fountain Inc. E-mail: ghorn@pulsent.com. M. Luby is with Digital Fountain, Inc. E-mail: luby@digitalfountain.com. M. Mitzenmacher is with the Division of Engineering and Applied Sciences at Harvard University and is supported in part by National Science Foundation operating grants CCR-9983832, CCR-0118701, CCR-0121154, and by an Alfred P. Sloan Research Fellowship. Part of the work was done while the author was visiting Digital Fountain, Inc. E-mail: michaelm@eecs.harvard.edu. W. Shaver is with the Oregon Institute of Technology and performed this research while interning at Digital Fountain, Inc. E-mail: shaver@qspeed.com. A preliminary version of this paper appeared in Proc. of NGC '00 in Stanford, CA [8].

Defining appropriate multicast congestion control algorithms which scale to large, heterogeneous audiences sizes is essential for enabling multicast "killer apps" such as reliable content distribution to large audiences [5] and video streaming [12]. Multi-rate congestion control, as opposed to single-rate congestion control [17], is a de facto requirement for scaling to large audience sizes, to avoid the problem of establishing a single session rate which caters to the receiver with the lowest end-to-end rate.

Standard approaches to multi-rate congestion control employ layered multicast [12], [18], [5] from a single source. Layered multicast organizes multiple multicast groups into logical layers. A host tunes its reception rate by subscribing to and unsubscribing from layers, i.e., by joining and leaving multicast groups. Different receivers may subscribe at different end-to-end subscription rates. A *cumulative* layered scheme has the additional property that all receivers must subscribe to and unsubscribe from layers in consecutive order. Several congestion control schemes for layered multicast sessions exist, but all have drawbacks.

Our congestion control scheme, which is based in part on the RLC protocol developed by Vicisano, Rizzo, and Crowcroft in [18], coexists with TCP, scales to large audience sizes, requires no changes to network routers or multicasting routing protocols, and faces no deployment hurdles (beyond those of deploying multicast in general). We call this scheme *FLID-DL*, which stands for "Fair Layered Increase/Decrease with Dynamic Layering", a preliminary description of which was given in [3]. FLID provides a generalization and simplification of RLC's TCP-like congestion control mechanisms for an arbitrary organization of multicast layers; DL is a strategy which effectively avoids a major response bottleneck caused by IGMP leave latency.

The Asynchronous Layered Coding (ALC) approach within the Reliable Multicast Transport (RMT) working group of the IETF provides a reliable layered multicast solution for content distribution [10]. ALC currently lacks a multi-rate congestion control protocol backed by consensus that is suitable for standardization within the IETF. We argue that FLID combined with DL provides a viable congestion control protocol for ALC.

The remainder of the paper is organized as follows. In Section II we discuss some of the basic issues associated with the design of multi-rate congestion control schemes and describe the RLC protocol in more detail. In Section III, we define DL and demonstrate its capability to eliminate the performance penalty of slow IGMP leave operations. In Section IV, we define FLID and show how it provides improved TCP-friendly congestion control. In Section V, we provide experimental evidence based on *ns* simulations to support our results, and we conclude with directions for future work in Section VI.

II. Related Work

There is a considerable body of work on multicast congestion control which we will not attempt to survey here; we refer the reader to the excellent recent survey article on the topic [19] and focus only on congestion control for *layered* multicast. The technique of congestion-controlled cumulative layered multicast was first proposed by McCanne, Jacobson and Vetterli [12] in the context of packet video transmission to large heterogeneous audiences. Their Receiver-Driven Layered Multicast (RLM) protocol achieves scalability by using a *receiver-driven* methodology, in which the hosts tune their subscription level by joining and leaving layers. They advocate an approach in which receivers periodically perform join experiments by subscribing to an additional layer and dropping a layer when they experience packet loss. There are several challenges that this approach introduces. First, one host's join experiments can introduce packet loss at other hosts behind the same bottleneck link, producing a potential source of unfairness or inefficiency. Second, standard approaches to cumulative layered multicast have exponentially increasing rates over the layers, which implies that the frequency of join experiments across the layers must be carefully designed to be friendly to TCP traffic and other sessions. Addressing these challenges motivated Vicisano, Rizzo, and Crowcroft to propose their Receiver-driven Layered Congestion Control (RLC) protocol [18].

A. RLC

RLC [18] was designed to provide a TCP-friendly multirate congestion control scheme which scales to large audience sizes, requires no modifications to routers or routing protocols, and does not require any coordination amongst receivers. For full scalability, a receiver-driven approach is required, as maintenance of per-receiver state at the source is infeasible and unscalable. But, uncoordinated join experiments by receivers pose substantial problems, as was observed in [12]. The authors of RLC cleverly avoid this problem by *synchronizing* join experiments. The source places synchronization points or increase signals into packets, where receivers can now only add a given layer after an appropriate increase signal for that layer. These increase signals are also *cumulative*, i.e., an increase signal j indicates that all receivers whose maximum subscription level is at most j can join a single additional layer. The use of cumulative increase signals solves the problem of synchronizing receivers behind a shared bottleneck, since when one receiver joins a layer that exceeds the bottleneck bandwidth, all other receivers behind that bottleneck will have also joined a layer. Then, since they will all experience packet loss, they will all drop back to their original rate prior to the join experiment.

In practice, care must be taken whenever a join experiment is performed, since by oversubscribing, a receiver can push the network into a state of congestion. To alleviate the congestion, the receiver must then unsubscribe from the layer by performing an IGMP leave operation, which can often incur substantial latency, leaving the network in a congested state[1]. Since oversubscription incurs a substantial cost, to minimize the likelihood of oversubscribing, the RLC source periodically injects a brief burst of packets on each layer prior to a synchronization point on that layer. The burst on layer i is designed to simulate the rate of layer $i+1$, the idea being that those receivers which lose packets during the burst learn that adding layer $i+1$ is unsafe, without incurring the cost of a join and leave operation. Unfortunately, if a receiver does *not* lose a packet in the burst, it still has no guarantee that adding the layer is safe, since the burst may be of insufficient length to induce packet loss (bursts recommended in [18] can be as brief as 8 packets). Thus a receiver is still prone to oversubscription. The complexity and lingering uncertainty associated with avoiding costly IGMP operations is one of the main problems with RLC which we address.

Another challenge addressed by RLC is the problem of appropriately orchestrating synchronization signals across the layers. The primary goal for RLC is to be fair to other instances of itself as well as to other congestion control algorithms such as TCP. As with most proposed layered multicast schemes, RLC requires that the rates on the layers must be exponentially spaced using a doubling scheme, i.e., the rates on the layers follow the pattern $1, 1, 2, 4, 8, \ldots$. While dropping a layer with this scheme performs a TCP-like multiplicative decrease, adding a layer suddenly doubles the rate. Therefore, RLC cannot be TCP-like at a fine granularity, since it cannot perform fine-grained additive increase. However, it performs TCP-like additive increase at a coarser granularity by placing increase signals on layer i at a frequency of $\frac{1}{R_i}$, where R_i is the cumulative rate through layer i. When used in conjunction with a doubling scheme on the layer rates, the trajectory induced by this distribution of increase signals corresponds to linear increase over large time scales.

One issue which RLC does not adequately address is the dramatic fluctuations in network bandwidth consumption and the potential for rapid queue buildup that a doubling scheme can induce. We recommend the use of schemes which exhibit slower exponential growth in the layer rate, providing gentler transitions during join experiments.

B. TCP fairness

An increasingly widely accepted measure of TCP friendliness is to compare the steady state throughput of a flow along a path to the throughput that TCP would achieve along that path. For TCP traffic, a great deal of work has been done to determine the equation expressing the throughput as a function of the packet size, the packet loss rate, and the round-trip time along that path [11], [8], [9], [14]. It has been advocated that any new congestion control algorithm should exhibit the same steady state flow rate as suggested by the TCP equation [11], [2]; ensuring that the flows will then share available bandwidth fairly across a bottleneck link, since across the bottleneck link

[1] We review the root causes of large IGMP leave latency in Section III-A.

both streams experience the same packet loss rate. In addition, there is an emerging body of research that suggests designing congestion control algorithms to explicitly use the TCP equation [9], [11], [15].

In order for a congestion control scheme to be fair in this sense against TCP, the flow rate must have the same behavior as the TCP equation over large time scales. (Note that while this is a necessary condition, it may not be a sufficient condition, as issues of variance and the mechanics of rate changes come into play. We will address this issue only through simulation.) The TCP throughput rate R, in units of packets per second, can be approximated by the formula in [14]:

$$R = \frac{1}{RTT\sqrt{q}(\sqrt{2/3} + 6\sqrt{3/2}q(1+32q^2))} \quad (1)$$

where R is a function of the packet loss rate q, the TCP round trip time RTT, and the round trip time out value RTO, where we have set $RTO = 4RTT$ according to [9]. In both RLC and our work, since different multicast hosts have different end-to-end latencies from the server, and since the multicast analogue of RTT is not well defined, a target value of RTT, which we call the *nominal round-trip time*, is fixed in advance to generate a target rate R.

C. Use of a digital fountain

Since receivers join and leave layers over time in layered multicast, it is hard to control or predict precisely which packets they will receive. While this is not particularly problematic for appropriately encoded streaming content [12] that does not need to be transmitted reliably, scheduling packets across the layers in *reliable* multicast applications is a challenging problem. Recently there has been much work on integrating forward error correcting (FEC) codes into layered multicast as an end-to-end solution for scaling reliable multicast to audiences with heterogeneous download bandwidths. The benefit of using an encoded data stream is that it is no longer necessary to solve the difficult problem of delivering every single packet to every single host, thus admitting some flexibility into the scheduling of data packets onto layers over time [18], [5]. In our implementation, we use a digital fountain encoding [5] to generate an effectively unbounded number of different forward error correcting packets to be scheduled among the different layers. In this approach, as soon as the receiver receives enough distinct encoding packets, it can recover the original data, independent of the particulars of which layers it subscribed to over time.

III. Dynamic Layering (DL)

A significant limitation of current approaches to congestion controlled layered multicast is the timeliness of joining and leaving groups. With all previous schemes, rate increases can only be accomplished by joining one or more multicast groups; likewise rate reductions can only be accomplished by leaving one or more groups. Large join latencies are not especially problematic, although they introduce sluggish behavior caused by rate increases occurring more slowly than anticipated. Large leave latencies pose severe problems however, as they limit responsiveness to congestion and can create unfairness to sessions which react relatively more quickly to congestion. In practice, IGMP leave latencies can in fact be very substantial, often on the order of several seconds.

A. IGMP leave latency

With the IGMP group membership protocol [6], [7], when a host wants to stop receiving content from a multicast group it sends a leave message to the last hop router. In general, the last hop router does not track the number of hosts beyond the interface participating in a given multicast group, thus it must poll the hosts to determine whether any are still active before stopping the flow of packets. To provide reliability, the router typically polls up to three times before terminating flow to the group. In current implementations, each polling attempt can take from 1 to 3 seconds, for an aggregate leave latency of between 3 and 9 seconds. During this time, multicast traffic continues to flow through the last-hop router, even if no subscribers are present. Therefore, deployable congestion control algorithms for layered multicast must avoid relying on IGMP leaves to respond to congestion effectively, at least until faster IGMP leaves [16] are implemented.

B. DL overview

The key feature we use to achieve this goal is the use of *dynamic* layers, or layers whose rates change over time. Dynamic layers are distinguished from *static* layers over which the rate of packet transmission to the layer remains fixed for the duration of the session. The use of static layers necessitates explicit IGMP leaves to perform congestion control; use of carefully designed dynamic layers does not.

Our approach employs the following paradigm: the sending rate of each layer decreases over time, thus a receiver can reduce its reception rate quickly, simply by not joining any additional layers. In order for receivers to maintain a given reception rate they must periodically join layers at a moderate pace, as though they are on a treadmill. In order to increase their reception rate they must join additional layers beyond those needed to maintain a constant rate. With this general approach, slow leave operations do not affect the responsiveness to congestion.

C. Emulating cumulative layered schemes

We now demonstrate how to emulate any static cumulative layer scheme with a dynamic layer scheme. Suppose there are ℓ static layers with rates $r_0, \ldots, r_{\ell-1}$ where zero is the index of the base layer, $\ell - 1$ is the index of the highest layer, and a receiver always subscribes to a cumulative set of layers starting from 0. Key parameters in designing the dynamic layer scheme are upper bounds on join and leave latencies.

We define J to be the worst case join latency and L to be the worst case leave latency. We assume that join latency is generally small (tens or hundreds of milliseconds) with small variance, while leave latencies can be much larger and much more highly variable. Now let s be an integer and T a real number satisfying $J < T < L < (s-1)T$. Our corresponding dynamic layer scheme uses $\ell + s$ dynamic layers. Each layer transmits at a fixed rate for a *time slot* of length T seconds.

Let $d_0, \ldots, d_{\ell+s-1}$ be the $\ell + s$ dynamic layers, and for convenience in describing the scheme, define $r_\ell = r_{\ell+1} = \ldots = r_{\ell+s-1} = 0$. In the dynamic scheme, the transmission rate on layer d_j has rate $r_{(\ell+j-i) \bmod (\ell+s)}$ during time slot i. An equivalent interpretation is that during time slot i, layer d_j carries the traffic corresponding to static layer $(\ell+j-i) \bmod (\ell+s)$. Hence, each layer d_j has a period of $\ell+s$ time slots, where it begins transmitting at the highest rate $r_{\ell-1}$, drops sequentially through rates $r_{\ell-2}$ down to r_0 at time slot boundaries, and then transmits no packets for s time slots. The periodicity described above is necessary to efficiently utilize the multicast address space, as each new layer corresponds to a separate multicast address.

Example 1: Consider a layering scheme where $\ell = 4$ and $r_i = i - 1$, for $i = 1, \ldots, \ell$. Let $J = 10$ ms and $L = 1.5$ seconds. If T is chosen to be 1 second, then $s = 3$. Figure 1 shows the rates on dynamic layers d_0 and d_1 for the first seven time slots.

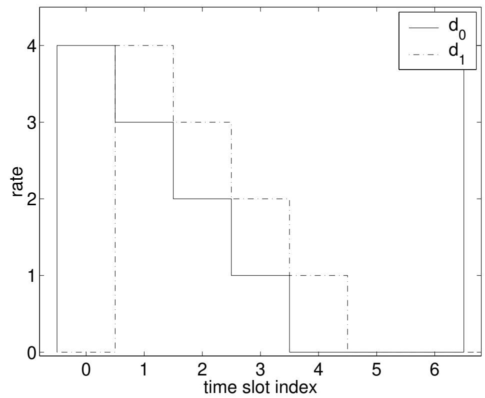


Fig. 1. The rates on dynamic layers d_0 and d_1 for the first seven time slots.

In order to emulate the behavior of a static layer scheme, there are three operations which a receiver could perform and which must be emulated on the dynamic layers. For clarity, we assume that we have a receiver currently subscribing to dynamic layers $d_{k-\ell}, \ldots, d_{k-\ell+i}$, i.e. emulating subscription to static layers $0, \ldots i$ in time slot k. (In the discussion of dynamic layering, all the indices are to be interpreted mod $\ell + s$.) The operations which a receiver can perform on the next time slot boundary are:

1. Emulate leaving static layer i. To do so, the receiver passively performs no action at the time slot boundary. The resulting aggregate rate will drop from $\sum_{j=0}^{i} r_j$ to $\sum_{j=0}^{i-1} r_j$.
2. Emulate retaining subscription to all current layers. To do so, the receiver joins dynamic layer $d_{k-\ell+i+1}$ at the time slot boundary.
3. Emulate joining static layer $i+1$: To do so, the receiver must join both dynamic layer $d_{k-\ell+i+1}$ and dynamic layer $d_{k-\ell+i+2}$ at the time slot boundary to move up to a rate of $\sum_{j=0}^{i+1} r_j$.

In all of the cases above, the receiver must also initiate a leave of dynamic layer $d_{k-\ell}$ at the time slot boundary. We emphasize that this leave need not complete quickly, as layer $d_{k-\ell}$ will transmit at a rate of zero for a substantial number of time slots. However, the leave must complete before the time slot at which reuse of layer $d_{k-\ell}$ begins. Indeed, this explains why we use $\ell + s$ layers; we allow a layer to transmit at zero rate for s consecutive time slots so that any leave message that occurred while the layer was transmitting has time to complete before the layer begins transmitting again at a non-zero rate. Note that when layer $d_{k-\ell}$ is reused, it will start transmitting at the maximum possible rate. This strategy also requires that the client know the current time slot (especially to correctly subscribe and leave dynamic layers). This can be accomplished in various ways, including using a separate multicast group for control information, or making the base layer static and embedding time slot information for the dynamic layers within the base layer.

To ensure that the reception rate of the receiver is smooth, a join can be scheduled just far enough in advance of the beginning of the time slot to ensure that packets start arriving from the joined dynamic layer just after the beginning of the next time slot. However, care must be taken that the join is not early enough to cause reception of packets from the joined dynamic group before the beginning of the time slot, as this may cause unnecessary congestion.

For the DL scheme, reactions (that is, joins and passive leaves) can occur at intervals of time T in a fast and predictable manner. The length of the time slot T is therefore a measure of the reactivity to network changes. T should be roughly the same as a small number of round trip times for TCP in order to be able to have approximately the same reaction times to changes in network conditions as TCP.

A receiver requires one leave and at most two joins to change its rate each T seconds. Smaller values of T make the system more responsive to loss; however, smaller values of T mean more join and leave requests per second as well as the need for more dynamic layers. The overhead for these operations in terms of bandwidth and control message overhead at last hop routers should be considered when designing the time slot and base layer bandwidth. Larger values of T mean that, on average, there is a longer time interval between the time at which packet loss occurs and the time at which the receiver can respond to congestion.

D. Emulating other layered schemes

While cumulative layering is the standard approach for layered multicast schemes, non-cumulative layering is also possible and advantages of such an approach are discussed in [4]. Dynamic layering can emulate a non-cumulative layering scheme, or indeed an *arbitrary* static layering scheme as follows. If the arbitrary static layering scheme consisting of ℓ layers, then we will emulate it with a dynamic scheme consisting of $s\ell$ layers, where s is as defined above. We will use s dynamic layers to emulate each static layer, cycling through those s dynamic layers so that only one is transmitting data during any time slot, and the transmission rate on the dynamic layer is the rate of the static layer (when it is not 0). Using s layers for each static layer allows receivers to leave a static layer without delay by leaving the appropriate dynamic layer; to remain subscribed to a layer, a receiver must repeatedly join the transmitting dynamic layer.

IV. Fair Layered Increase/Decrease

Fair Layered Increase/Decrease (FLID) is a protocol that is used to allow receivers to increase and decrease their reception rates based on congestion conditions in a manner that is similar to the TCP-friendliness equations for a TCP flow with a fixed RTT value. FLID can either be implemented on top of DL, or on top of a static layering scheme (preferably in an environment where leave operations do not incur large latency).

FLID is akin to RLC [18] in several ways. The server places signals into packets that completely dictate the behavior of receivers with regards to joining and leaving layers. This property helps to coordinate the behavior of receivers behind bottleneck links. Like RLC, there is no feedback from receivers to the server, and different receivers may join different numbers of layers depending on the different network conditions on the paths between the server and the receivers. These properties make FLID scalable to an unlimited number of receivers. In particular, receivers with slower bandwidth connections to the server do not slow down receivers with faster connections.

We introduce the FLID congestion control algorithm in three parts. First, we discuss the methods we employ to set the rates of the different layers. Next we present the actual rules for adding and dropping layers, and finally, we discuss how to set the increase signals at the server.

A. Rates on the different layers

FLID uses a cumulative layered scheme on ℓ layers. When a receiver subscribes to a set of layers $0, 1, \ldots, i$, where $i \leq \ell - 1$, we call i the *subscription level*, or simply the level of the receiver. Generally the first layer is called the *base layer*, which in our case will be a static layer. To start receiving traffic for a multicast session, the host joins the base layer.

As before, r_i is the rate, in packets per second, for layer i and $R_i = \sum_{j=0}^{i} r_i$ is the cumulative rate, in packets per second, for layers 0 through i, i.e., the reception rate for a receiver with a subscription level i.

There are a variety of methods for choosing the different rates for the ℓ different layers. Some examples of schemes include:

- Equal scheme (E): The rates for all layers are equal, e.g., 2.5 packets per second. For example, with $\ell = 20$ layers and a 1 KB packet size, the subscription level can range from 20 Kbps up to 400 Kbps.
- Increase by one scheme: The relative increases in the rates for the layers are in the sequence $1, 2, 3, 4, \ldots$, i.e., $r_{i+1} = r_i + 1$. For example, with a base layer rate of $R_0 = 2.5$ packets per second, $\ell = 20$ layers and a 1 KB packet size, the subscription level can range from 20 Kbps up to 4.2 Mbps.
- Doubling scheme (D) [18]: Adding another layer doubles the subscription level. The relative increases in the rates for the layers are in the sequence $1, 1, 2, 4, 8, 16, 32, 64, \ldots$, i.e., $R_{i+1} = 2R_i$. For example, with a base layer rate of $R_0 = 2.5$ packets per second, $\ell = 20$ layers and a 1 KB packet size, the subscription level can range from 20 Kbps up to 10 Gbps.
- Multiplicative scheme (M): A generalization of the doubling scheme where the rate for subscription level i is proportional to c^i for a fixed constant $c > 1$. The relative increases in the rates for the layers are in the sequence $1, c-1, c^2-c, c^3-c^2, c^4-c^3, \ldots$, i.e. $R_i = c^i R_0$. For example, with a base layer rate of $R_0 = 2.5$ packets per second, $\ell = 20$ layers, a 1 KB packet size and $c = 1.3$, the subscription level can range from 20 Kbps up to 2.9 Mbps.
- Heavy-tail scheme: Set the rates on the layers so that if k independent FLID schemes are sharing a common bottleneck, then they each obtain a $1/k$ share of the bottleneck rate C. The base layer rate is chosen to be C/ℓ. For layer i, $r_i = C/((\ell - i)(\ell - i - 1))$, so $R_i = C/(\ell - i)$. For example, with a bottleneck link rate of $C = 1$ Mbps and $\ell = 20$ layers, the rates for the layers in Kbps are 51.2, 2.7, 3.0, 3.3, 3.8, 4.3, 4.9, 5.6, 6.6, 7.8, 9.3, 11.4, 14.2, 18.3, 24.8, 34.1, 51.2, 85.3, 170.7, 512.
- Tailored to specific subscription levels: Set the rates on the layers so that the subscription level can be adjusted to be close to the capacity of the bottleneck link. For example, suppose there are different hosts behind bottleneck links that have capacities of 28 Kbps (modem), 128 Kbps (ISDN), 500 Kbps (low end DSL), 1.5 Mbps (mid range DSL, T1, cable), 10 Mbps (10 Mbps Ethernet, fast DSL, fast cable) and 100 Mbps (100 Mbps Ethernet) respectively. Suppose further that for each bottleneck, a subscription level of 80% and 90% of capacity is desired. Here it may be desirable to put in a subscription level for each bottleneck that is just over the capacity, say 110% of capacity, so that when this subscription level is attempted, the host feels loss but the subscription level does not jump up dramatically to the next bottleneck link rate.

We focus on the multiplicative scheme for the remainder of the paper, for the following reason. Two useful factors to consider in evaluating a cumulative layered multicast scheme are the number of layers ℓ needed to span a given range of reception rates and the granularity with which a receiver can tune its rate within that range. In general, the tradeoff is that the larger the value of ℓ, the more fine-grained rate changes can be and the smoother the reactions of the congestion control algorithm, but the more layers and hence multicast addresses that are needed for the transmission to achieve the same range of cumulative rates.

The multiplicative scheme has the desirable property that it uses a number of layers ℓ that is logarithmic in the total range of reception rates, which seems important for scalability, as it is currently infeasible and undesirable to employ a large number of multicast groups to satisfy receivers of a single layered multicast session. At the same time, using a multiplicative factor of c that is less than two allows a more fine-grained rate adjustment. Further discussion about performance measures for various layering schemes can be found in [4].

We emphasize that although we focus on the multiplicative layering scheme hereon the FLID approach can be applied to the other layering schemes described above.

B. Increase and Decrease Rules

In FLID, the server partitions time into slots of duration T seconds each. All packets transmitted by the server in each time slot include the current *time slot index*. From the receiver point of view, a new time slot starts when the first packet is received with a new time slot index.

Time slots are used to coordinate the activities of receivers. If during a time slot a receiver measures any packet loss, the receiver must decrease its subscription level by one at the end of the time slot. The receiver ignores all subsequent packet losses until the end of the time slot, i.e., the receiver only considers a single congestion event per time slot.

Time slots are also used to coordinate actions when receivers increase their subscription level. The server places an *increase signal* into each packet. The increase signal identifies a subscription level which is fixed for all packets on all layers within a time slot. Receivers use increase signals to decide whether to increase their subscription level at the beginning of a time slot according to the following rule: if the current subscription level is i, the increase signal for the current time slot is j where $j \geq i$, and there is no packet loss during the current time slot, then increase the subscription level by 1 at the beginning of the next time slot. An increase signal of -1 is used to indicate that no receiver should increase its rate.

We use cumulative increase signals for the following reasons:
1. If a receiver behind a bottleneck link adds a new layer and does not experience congestion, then receivers sharing the same bottleneck link at lower subscription levels should also add a layer to fully exploit the available bandwidth;
2. If a receiver adds a new layer and causes congestion on a bottleneck link, other receivers at a lower subscription level behind this bottleneck should not be punished. If all low-rate receivers raise their subscription level in tandem with a higher-rate receiver, the worst that can happen is that the low-rate receivers drop back to their previous rate if congestion occurs.

Of course there should be more increase signals for receivers at lower subscription levels than for those at higher subscription levels so that eventually all receivers behind the same bottleneck link reach the same subscription level. The pattern of increase signals that the server uses is one of the more important considerations in the design of FLID. This pattern is designed in conjunction with the rates of the different layers, and together they determine the fairness of FLID against other protocols such as TCP.

C. Setting the increase signals for the layers

To gain insight into how to set the increase signal values we consider a probabilistic pattern of increase signals. We then describe a deterministic pattern. Let $1 = p_{-1} > p_0 > p_1 > \ldots > p_{\ell-1} = 0$. In each time slot, the increase signal is set to i with probability $p_i - p_{i+1}$ for $-1 \leq i \leq \ell-1$. This means that a receiver with subscription level i will increase its level in each time slot with probability p_i. The value of p_{l-1} must be set to zero, to prevent a receiver joined to all layers from ever attempting to join an additional layer that does not exist. For simplicity, we define $p_{-1} = 1$, which corresponds to a phantom layer that carries no traffic.

In the description that follows, we assume that each packet is lost independently with a fixed probability q. This is in fact the model used to derive the TCP throughput equation (1), and provides a suitable and representative example of the results we can achieve. Our approach can be used in significantly more general settings, however. To do so, one first must be given a loss model, and a means of determining the behavior of FLID under the loss model. Often, if the model is sufficiently simple, this can be done analytically, as we do below; otherwise it can be accomplished via simulation. Second, one must be given a target TCP behavior; here we use long-term throughput versus the loss probability q. The problem then becomes to choose the p_i values so that FLID is as close as possible to the target behavior. This is just a multi-dimensional optimization problem, for which standard techniques may be applied. (We suggest a hill-climbing approach in the appendix.) It would therefore be relatively straightforward to use our approach to handle more complex loss models, such as simple Markovian burst loss models.

Again, in what follows, we assume that each packet is lost independently with a fixed probability q. Define $P_j(k|i)$ to be the probability of a receiver moving from subscription level i to subscription level k at the beginning of time slot j. Since we can only increase or decrease the subscription level by a single layer at a time, $P_j(k|i) = 0$ for $|k - i| > 1$. In a time slot of length T, a receiver with subscription level i receives $T \cdot R_i$ packets, so we have

$$P_j(k|i) = \begin{cases} 1 - (1-q)^{T \cdot R_i} & k = i - 1 \\ (1 - p_i)(1-q)^{T \cdot R_i} & k = i \\ p_i(1-q)^{T \cdot R_i} & k = i + 1 \\ 0 & \text{otherwise} \end{cases}$$

The transition probability $P_j(k|i)$ is independent of the time slot j so we write the transition probability as $P(k|i)$ (also under the assumption that q does not vary over time).

The receiver now behaves like an ℓ state Markov process, a part of which is shown in Figure 2. We denote by f_i the steady-state fraction of the time the receiver will have a subscription level of i. At steady state, the average

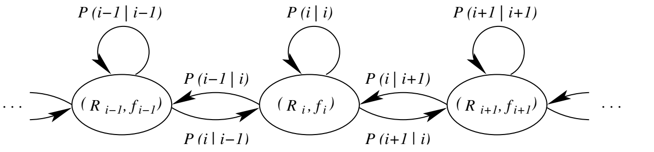


Fig. 2. The Markov process showing the transitions for state (R_i, f_i).

throughput $\hat{R}$ is then

$$\hat{R} = \sum_{i=0}^{\ell-1} f_i R_i. \tag{2}$$

By the structure of the state diagram, at steady state the flow into each state on an edge has to be equal to the flow out of that state on that edge, i.e.,

$$P(i+1|i) \cdot f_i = P(i|i+1) \cdot f_{i+1}, \tag{3}$$

or

$$f_{i+1} = \frac{(1-p_i)\,(1-q)^{T \cdot R_i}}{1-(1-q)^{T \cdot R_{i+1}}}\ f_i \tag{4}$$

for $i = 0, \ldots, l-2$. Therefore, given a set of p_i and R_i values, it is simple to calculate $\hat{R}$ using the recursion in (4) and equation (2).

To match the TCP throughput in 1, we have the following problem: given the target functional relationship between $\hat{R}$ and q, find a set of strictly decreasing p_i values that approximate this function. In the appendix, we present an intuitive argument for setting the p_i values heuristically. We then develop a more sophisticated technique based on a hill-climbing algorithm.

Note also that the above analysis also immediately tells us the rate of join operations. Recall that each time slot requires a single leave operation and either 0, 1, or 2 join operations depending on whether the subscription level decreases, stays the same, or increases, respectively. From the above equations for $P(k|i)$, the expected number of joins per time slot when at subscription level i is $(1+p_i)(1-q)^{T \cdot R_i}$, and hence the steady state rate of join operations is $\sum_i f_i(1+p_i)(1-q)^{T \cdot R_i}$.

Example 2: Consider a base layer of $R_0 = 3$ packets per second, i.e., 24 Kbps for 1 KB packets, $\ell = 30$ and a multiplicative layering scheme with $c = 1.3$. For a time slot of $T = 0.5$ seconds and a nominal round-trip time of 0.1 seconds, if we set the p_i values according to the hill-climbing algorithm in the appendix, then we have the expected number of joins and leaves per second versus the packet loss rate q shown in Figure 3.

We emphasize that while we have techniques to match TCP throughput extremely closely, and can therefore make FLID fair to TCP streams in this sense, it is not necessarily clear that is is desirable to do so. Since the FLID session may be serving many users, some systems may prefer to make FLID reasonably fair to TCP streams, while still giving FLID sessions some advantage. For example, perhaps it would be sufficient for FLID throughput to be within a specified constant factor of TCP throughput over some range of loss probabilities. The hill-climbing techniques we describe have the advantage that they can be used to match any desired throughput curve. Also, we emphasize that this simplified analysis ignores many issues that may affect performance in practice, such as buffer sizes, the speed of reaction to congestion, and the granularity of FLID layers. Hence simulations are valuable in obtaining a better understanding of performance of a given FLID setting in practice.

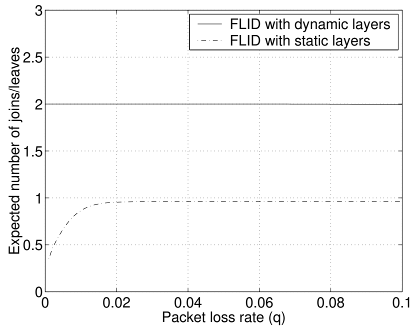


Fig. 3. The expected number of joins and leaves per second versus the packet loss rate q for FLID with static layers and dynamic layers.

C.1 Use of a reverse binary counter

As mentioned previously, we use a deterministic scheme to set the increase signal in each time slot. This is because if we choose the increase signal in each time slot at random, then we occasionally increase the receive rate too rapidly when a receiver obtains several increase signals over a short time interval. Our deterministic scheme still generates an increase signal that allows a receiver to increase its subscription level from i to $i+1$ about every $1/p_i$ time slots, but it has the additional property that the variance in the number of time slots between increase signals for each layer is minimal. We note that under the assumption that losses are independent, the steady-state probabilities of the Markov process will be the same for this deterministic scheme as for the probabilistic one, so our throughput remains compatible with TCP.

Our scheme uses a reverse binary counter. Let

$$b = b_0 b_1 \ldots b_{s-1}$$

be a number written in binary notation, i.e., b_i is a single bit and b is a concatenation of s bits. Let

$$\hat{b} = 0.b_{s-1} b_{s-2} \ldots b_1 b_0$$

indicate b written backwards interpreted as a real number in the range $[0, 1]$.

Example 3: If $b = 010011$, then $\hat{b} = 0.110010$, i.e., the real number 0.78125 written in decimal notation.

At the beginning of a session, set b to be an arbitrary value, then, at each time slot, the algorithm for choosing the increase signal is to increment b by 1, find the largest layer i

such that $p_i \geq \hat{b} \geq p_{i+1}$, then set the increase signal to i. We increment b modulo 2^s so the counter is reset every 2^s time slots.

Example 4: Suppose there are four layers with

$$p_0 = 0.33,\ p_1 = 0.21,\ p_2 = 0.09 \text{ and } p_3 = 0.$$

If b is initially set to 1, then the values of b for the first 8 time slots are shown in the first column of Table I. The corresponding values of $\hat{b}$ are shown the second and third columns, written in binary and decimal notation respectively. Finally, the increase signal layers for the first 8 time slots are shown in the last column.

Time Slot	b	$\hat{b}$		Increase Signal
1	0001	0.1000	0.5	-1
2	0010	0.0100	0.25	0
3	0011	0.1100	0.75	-1
4	0100	0.0010	0.125	1
5	0101	0.1010	0.625	-1
6	0110	0.0110	0.375	-1
7	0111	0.1110	0.875	-1
8	1000	0.0001	0.0625	2

TABLE I

THE VALUES FOR b AND $\hat{b}$ FOR THE FIRST 8 TIME SLOTS AND THE CORRESPONDING VALUES FOR THE INCREASE SIGNAL FOR $p_0 = 0.33,\ p_1 = 0.21,\ p_2 = 0.09$ AND $p_3 = 0$.

In this example, in the second time slot, only a receiver with subscription level 0 can increase its rate. A receiver with subscription level 0 or 1 can increase its rate in the fourth time slot and a receiver with subscription level 0, 1 or 2 can increase its rate in the eighth time slot. In all other time slots, no receiver may increase its rate.

V. Experiments

In order to study the suitability of FLID-DL for Internet deployment, we examine its behavior extensively using *ns-2* [13]. We demonstrate two sets of experiments. In the first set, we consider how to set the various parameters of FLID-DL and show how FLID-DL addresses some of the shortcomings of RLC. In the second set, we examine how FLID-DL scales, measure its behavior to a set of heterogeneous clients, and demonstrate that it coexists fairly with different types of TCP traffic. We present only a summary of our findings here and refer the reader to the FLID website at `www.digitalfountain.com/technology/library/flid` for further details.

A. Setting the FLID-DL parameters

There are three parameters that we need to set for FLID-DL:
1. the rates on the different layers;
2. the nominal round-trip time; and
3. the duration of the time slots.

We implement FLID in our simulations with a multiplicative layering scheme to set the rates on the different layers. We use a base layer of 24 Kbps and a packet size of 1 KB, i.e., $R_0 = 3$ packets per second. For each independent FLID session we pick the number of layers ℓ large enough so that the highest reception rate possible for each individual session is greater than the capacity of the bottleneck link, up to a maximum of 30 layers.

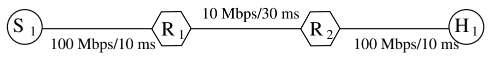


Fig. 4. The network topology used to set the multiplicative factor c on the different layers. The nodes on the left are servers, the nodes on the right are hosts, and the horizontal link in the middle is the bottleneck link.

To choose the rates on the different layers, we vary the value of the multiplicative factor c from 1.2 to 2.0. For each c value, we simulate a single FLID-DL session for 100 seconds and measure the throughput and number of packets lost over the final 50 seconds. Our topology consists of one server and one host connected by two drop-tail routers, with a queue size of 128, and a 10 Mbps bottleneck between the two routers as shown in Figure 4. The time slot duration is 500 ms and the increase signals are chosen according to the 120 ms round trip time of the topology.

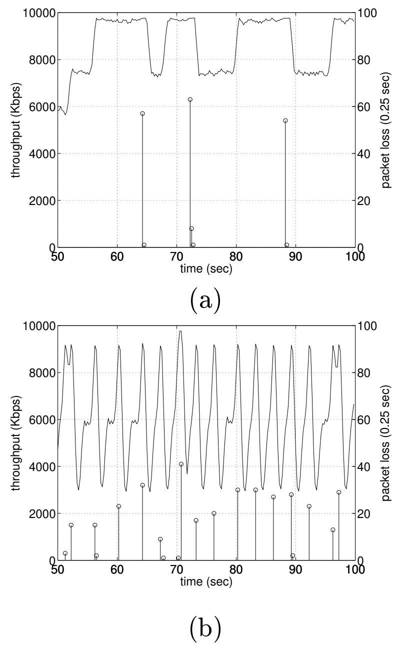


Fig. 5. The throughput (solid line) and number of lost packets (circles) for a single FLID-DL session with (a) $c = 1.3$ and (b) $c = 2.0$.

Figure 5 shows the throughput and number of packets lost in each 250 ms interval for $c = 1.3$ and $c = 2.0$. For $c = 1.3$, the bandwidth utilization is 85%, and an average of 47 packets are dropped per packet loss event. For $c = 2.0$, we had a bandwidth utilization of 59%, and an average of

17 packets dropped per packet loss event. However, over the 50 second interval, the latter experiment lost 2.5 times as many packets, as the large jumps in transmission rates cause more frequent packet loss.

Based on extensive simulations, of which this experiment is one representative, we chose $c = 1.3$ as our ideal multiplicative factor, since it is large enough to allow us to use few layers, but small enough to give us a sufficiently small granularity in the subscription level to avoid abrupt rate increases. It also gives a reasonably small packet loss rate when we exceed the bottleneck bandwidth. With this choice of c, FLID-DL is somewhat less responsive to congestion than TCP, since our multiplicative decrease is smaller than that of TCP. We have not found this to result in significant unfairness in our simulations; for more on the impact of slowly-responsive congestion control we refer the reader to [1].

We choose our nominal round-trip time to be 100 ms, so that FLID-DL competes fairly with TCP with an RTT of 100 ms. We set the ratio of the time slot duration to the nominal round-trip time using the same topology, except we now have random independent packet loss of 4% on the bottleneck link, i.e., we have no loss due to the queues overflowing. For a fixed nominal round-trip time, we varied the time slot duration and measured the throughput. For a given packet loss rate, doubling the time slot duration requires that we halve the reception rate in order to achieve the same probability of a packet being lost in each time slot. In order to achieve a reasonable reception rate; avoid large packet burst loss; and avoid a large number of join/leave operations per second, we chose the time slot duration to be 500 ms, or a factor of 5 larger than the nominal round-trip time used in our subsequent experiments. When simulating multiple TCP connections over a single bottleneck link in an event-driven simulator, synchronization problems can arise. To prevent this we start our connections at slightly different times by adding a small random delay to simulate processing overhead before each packet is transmitted.

B. Static vs. dynamic layered schemes

Our next experiment compares the behavior of four independent FLID-DL sessions to four independent FLID-SL (static layer) sessions, when we have a random leave latency uniformly distributed between 2 and 4 seconds. Our topology consists of a simple double star network as shown in Figure 6. Both routers are drop-tail with a queue size of 50 packets.

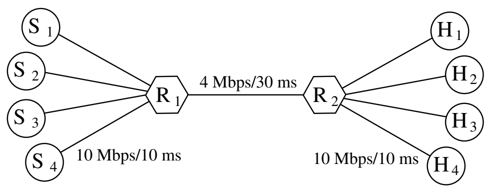


Fig. 6. The network topology used to compare the static vs. dynamic layered schemes.

Figure 7 shows the behavior of four FLID-DL sessions and four FLID-SL sessions with a leave latency of between 2 and 4 seconds. The FLID-DL sessions have a total bandwidth utilization of 90%, while for FLID-SL the bandwidth utilization is only 78%. In fact these results overstate the performance of FLID-SL, in that FLID-SL has reasonable bandwidth utilization only because, in this simulation, hosts continue to accept packets received on groups from which they had unsubscribed, but for which the last-hop router still continued to forward packets. The FLID-SL sessions combined lose three times as many packets as their FLID-DL counterparts.

We have also run the same experiment with FLID-SL with various other leave latencies. When the IGMP leave latency is zero, we find that the behavior of FLID-SL and FLID-DL are very similar. The results are also similar when we used RED routers setting the queue size to 100 packets, set *maxthresh* to 50, *minthresh* to 5, and turned the *gentle* setting on.

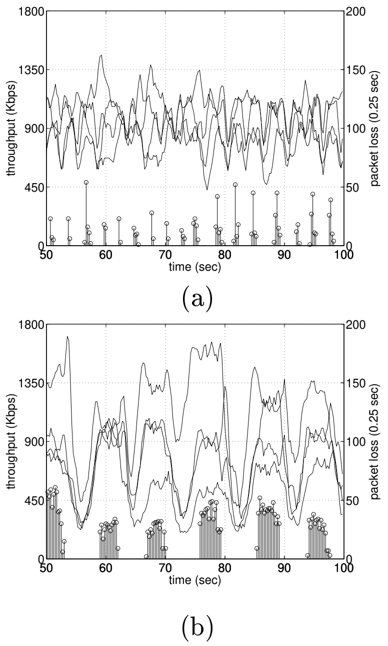


Fig. 7. The throughput (solid line) and number of lost packets (circles) for (a) 4 FLID-DL sessions and (b) 4 FLID-SL sessions with a 2 to 4 second leave latency.

C. Coordination behind a bottleneck link

To demonstrate how the hosts coordinate behind a common bottleneck link, we have 100 hosts subscribe to the same FLID-DL session at random times chosen uniformly between 0 to 5 seconds. We use the same topology as in Figure 6 except now with 100 hosts. Figure 8 shows that all 100 hosts converge to the same subscription level after 28 seconds. A number of other experiments demonstrating coordination of hosts that arrive asynchronously behind a shared bottleneck are provided on the FLID-DL website. In the scenarios we have considered, convergence time is comparable to that depicted here.

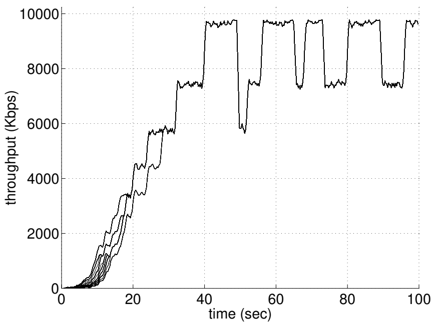


Fig. 8. The coordination of 100 hosts behind a bottleneck link.

D. Random loss and heterogeneous delays

We test the scalability of FLID-DL in the presence of loss, where we have a number of hosts subscribed to a single FLID-DL session. We generate random loss at various points on each topology and measure the throughput downstream at the hosts. We find that the throughput of each FLID-DL host depends only on the loss rate experienced by that particular host. Experimental evidence indicates that FLID-DL scales well in the presence of random loss.

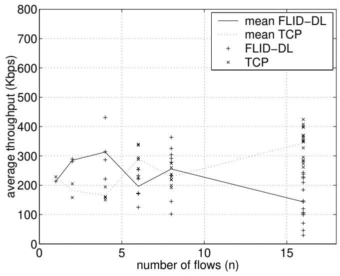
When we vary the delay of each host behind a common bottleneck, we find that the throughput at each host is proportional to both the nominal round-trip time and the random loss rate and is reasonably independent of the delay experienced by each host (except for the fact that the join at the highest layer experiences the longest latency). When the delay is greater than the time slot duration, FLID-DL has a subscription level that is higher than the actual reception rate. Since FLID-DL reacts to increase signals for its subscription level, as opposed to its actual reception rate, FLID-DL behaves less aggressively as the delay increases to the order of a time slot.

E. TCP fairness

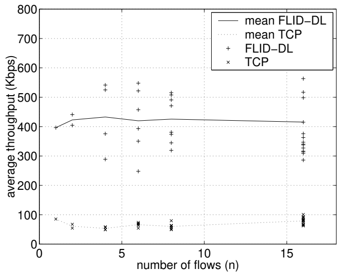
For our final experiment, we compare how FLID-DL competes with TCP Reno and TCP SACK. We have n TCP and n FLID-DL streams share a common bottleneck of $0.5n$ Mbps. We vary n and calculate the throughput for the final 50 seconds of a 100 second simulation.

Figure 9(a) shows the relative throughput of FLID-DL and TCP for a drop-tail queue of size $7n$ packets. Each point in the graphs represents the throughput of an individual stream. We also show the mean throughput for each type of stream. In this figure, which is representative of a large number of additional simulations, FLID-DL and TCP share the available bandwidth equitably. Figure 9(b) shows the relative throughput of FLID-DL and TCP when we increase the drop-tail queue size to $25n$ packets. In this scenario, FLID-DL is unfair to TCP and the average FLID-DL flow achieves a throughput 4 to 8 times larger the average TCP flow.

FLID-DL becomes less fair to TCP as the queue size increases, since it does not adjust its reception rate in response to changes in the network round trip time. In contrast, a TCP flow responds to the increased latency introduced when a queue fills by reducing its rate of additive increase. Because of this, in an environment where variable queueing delays can account for a significant portion of the end-to-end latency, FLID-DL will not be fair to TCP. Of course this effect could be ameliorated by changing FLID parameters, such as the nominal round-trip time. The experiment does suggest that achieving fairness to the extent that FLID-DL and TCP realize nearly identical throughput over a wide variety of network conditions may not be realizable. Nevertheless, our experiments reveal that FLID-DL can be made to be roughly fair to TCP, in that the throughput achieved is within a small constant factor of TCP streams sharing the same end-to-end path, across a large spectrum of situations.



(a)



(b)

Fig. 9. The fairness of FLID-DL and TCP Reno for a queue size of (a) $7n$ and (b) $25n$.

VI. Conclusion

We have demonstrated that the use of dynamic layering admits an elegant solution to large IGMP leave latencies, without requiring changes to IGMP, routers, or other multicast routing protocols. We have also outlined the FLID scheme, which generalizes the RLC protocol to accommodate a wide spectrum of multiplicative increase rates but eliminates some of its complexity, such as the need for bandwidth probes. DL combined with FLID provides a significant step towards a complete and scalable receiver-driven congestion control algorithm for layered multicast. We hope that our experiences with FLID-DL will encourage additional interest and work in multi-rate multicast congestion control.

Acknowledgements

The authors would like to thank Mark Handley and Lorenzo Vicisano for their feedback and their help in compiling a list of suitable experiments to test the FLID-DL protocol. We also thank Don Towsley, the anonymous NGC '00 reviewers and the anonymous J-SAC referees for their suggestions for improving this paper.

References

[1] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms. In *Proc. ACM SIGCOMM*, pages 263–273, August 2001.

[2] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K.K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet, IETF RFC 2309, April 1998.

[3] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver. FLID-DL: Congestion Control for Layered Multicast. In *Proceedings of the Second Int'l Workshop on Group Communication (NGC '00)*, pages 71–81, Stanford, CA, November 2000.

[4] J. Byers, M. Luby, and M. Mitzenmacher. Fine-Grained Layered Multicast. In *Proc. IEEE INFOCOM*, pages 275–283, April 2001.

[5] J. Byers, M. Luby, and M. Mitzenmacher. A Digital Fountain Approach to Asynchronous Reliable Multicast. In *IEEE Journal on Selected Areas in Communication*, 2002. A preliminary version of this paper entitled "A Digital Fountain Approach to Reliable Distribution of Bulk Data" appeared in *Proc. ACM SIGCOMM*, pages 56-67, Vancouver, Canada, September 1998.

[6] S. Deering. *Multicast Routing in a Datagram Internetwork.* PhD thesis, Stanford University, Stanford, CA, December 1991.

[7] W. Fenner. Internet Group Management Protocol, Version 2. IETF RFC 2236, January 1997. Available at `ftp://ftp.isi.edu/in-notes/rfc2236.txt`.

[8] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Trans. on Networking*, August 1999.

[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. ACM SIGCOMM*, pages 43–56, August 2000.

[10] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft. Asynchronous Layered Coding protocol instantiation. IETF Internet Draft draft-ietf-rmt-pi-alc-06.txt, February 2002, expires August 2002, work in progress.

[11] J. Mahdavi and S. Floyd. TCP-friendly Unicast Rate-based Flow Control. Note sent to end2end-interest mailing list, January 1997.

[12] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proc. ACM SIGCOMM*, pages 117–130, Stanford, CA, August 1996.

[13] ns: Network simulator. Available at `http://www.isi.edu/nsnam/ns`.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, April 2000.

[15] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Real-time Streams in the Internet. In *Proc. IEEE INFOCOM*, pages 1337–1345, March 1999.

[16] L. Rizzo. Fast Group Management in IGMP. In *Hipparch Workshop*, pages 32–41, London, June 1998.

[17] L. Rizzo. pgmcc: A TCP-friendly single-rate multicast congestion control scheme. In *Proc. of SIGCOMM 2000*, pages 17–28, Stockholm, August 2000.

[18] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like Congestion Control for Layered Multicast Data Transfer. In *Proc. IEEE INFOCOM*, pages 996–1003, San Francisco, CA, March 1998.

[19] J. Widmer, R. Denda, and M. Mauve. A Survey on TCP-Friendly Congestion Control. *IEEE Network*, 15:28–37, May 2001.

Appendix

A. Heuristic for setting the p_i values

We now provide a sketch of the method for how to set the FLID parameters so that it behaves according to the TCP equation in the face of a fixed packet loss rate q. We model packet loss by a process where each packet is lost independently with probability q, which is in fact the model used to derive the TCP equation (1).

Suppose the current subscription level is i, so that the aggregate rate is R_i. Since the probability that each packet is lost is q, on average there are roughly $t_d = 1/(q\,R_i)$ seconds between packet loss events. On the other hand, the increase signal when the subscription level is i occurs on average each $t_u = T/p_i$ seconds. If $t_d = t_u$, then the rate is as likely to go up as down for subscription level i. This occurs when $1/(q\,R_i) = T/p_i$, i.e., when

$$p_i = TqR_i. \tag{5}$$

We use this to set the values for the p_i's since this approximately equates the probability of increasing and decreasing the rate in FLID, according to our Markov chain description in Section IV-C. We would expect that if the steady state rate is R_i, then the rate is roughly as likely to go up as it is to go down. While this is not precisely true since it depends on the relative values of the R_i's, it is a good first approximation. We then set the p_i values as follows:

1. Choose an appropriate nominal round-trip time.
2. For each subscription level R_i, solve the TCP equation (1) for $q = q_i$ by setting R to R_i and RTT according to step 1.
3. Set $p_i = \min\{Tq_iR_i, 1\}$ for $i = 0, \dots \ell - 1$.

When solving the TCP equation, the values of p_i may not be monotonically decreasing due to the influence of the RTO value on the TCP equation for large loss rates. To ensure that the FLID rules are followed, the p_i values for the small layers can be set to the maximum p_i value.

Example 5: Consider a base layer of $R_0 = 3$ packets per second, i.e., 24 Kbps for 1 KB packets, $\ell = 30$ and a multiplicative layering scheme so the subscription level R_i is equal to $R_0\,c^i$. We set $T = 0.5$ seconds and the nominal round-trip time to be 0.1 seconds.

For $c = 1.3$, the explicit (i, R_i, p_i) triples are:

(0, 24.0 Kbps, 0.932)	(15, 1.20 Mbps, 0.441)
(1, 31.2 Kbps, 0.932)	(16, 1.56 Mbps, 0.353)
(2, 40.6 Kbps, 0.932)	(17, 2.03 Mbps, 0.278)
(3, 52.7 Kbps, 0.932)	(18, 2.64 Mbps, 0.217)
(4, 68.5 Kbps, 0.932)	(19, 3.43 Mbps, 0.169)
(5, 89.1 Kbps, 0.932)	(20, 4.45 Mbps, 0.131)
(6, 116 Kbps, 0.932)	(21, 5.79 Mbps, 0.100)
(7, 150 Kbps, 0.932)	(22, 7.53 Mbps, 0.078)
(8, 196 Kbps, 0.932)	(23, 9.79 Mbps, 0.060)
(9, 255 Kbps, 0.932)	(24, 12.7 Mbps, 0.046)
(10, 331 Kbps, 0.899)	(25, 16.5 Mbps, 0.035)
(11, 430 Kbps, 0.840)	(26, 21.5 Mbps, 0.027)
(12, 559 Kbps, 0.751)	(27, 27.9 Mbps, 0.021)
(13, 727 Kbps, 0.649)	(28, 36.3 Mbps, 0.016)
(14, 945 Kbps, 0.541)	(29, 47.2 Mbps, 0.0)

For $c = 1.3$, the graphs of the functions plotting average throughput (as derived from the TCP equation (1) and the FLID equation (2)) versus loss rate for TCP and FLID with these settings is given in Figure 10. As can be seen, the curves are comparable. We obtained similar results for other values of c provided that the ratio of the time slot duration to the nominal round-trip time was not larger than a factor of about 20 to 1.

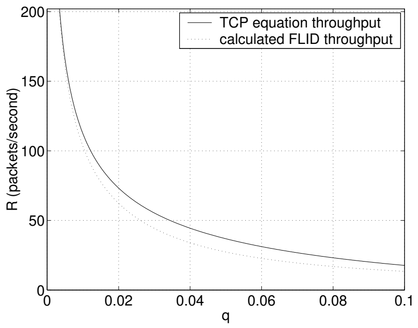


Fig. 10. Comparison of the estimated average throughput as a function of loss probability for FLID and TCP where the p_i values are obtained using the heuristic.

B. Hill-Climbing Algorithm

The previous heuristic yields a set of p_i values that closely approximate the behavior of the TCP equation. However, because the analysis is clearly only approximate, we now describe an alternative approach that allows us to find a set of p_i values that provide a closer match, if this is desired, by applying a simple hill-climbing algorithm. Our approach here is more general in that it applies to any set of points describing the throughput for a given set of packet loss event rates.

A hill-climbing algorithm requires that we apply some metric to the p_i values as a measure of improvement. For this, we consider a specific range of loss rate probabilities $[q_a, q_b]$. We choose h linearly or logarithmically spaced points $q_a = q_1, q_2, \ldots, q_h = q_b$, and evaluate the TCP equation (1) to determine the steady state transmission rate at these points for a given RTT. In fact we can evaluate any throughput equation for these loss rates. We denote the throughput versus packet loss rate curve as $R(q)$.

Given a set of p_i values we proceed as follows. For each q_j, $j = 1, 2, \ldots, h$, we compute the average throughput $\hat{R}(q_j)$ at steady state by solving the Markov chain using the recursion in (4) and equation (2) in Section IV-C. We denote the resulting throughput versus packet loss rate curve as $\hat{R}(q)$. We then calculate the normalized mean square error ($NMSE$) between $\hat{R}(q)$ and $R(q)$ as follows,

$$NMSE(\hat{R}(q), R(q)) = \sum_{j=1}^{h} \frac{(\hat{R}(q_j) - R(q_j))^2}{R(q_j)^2}.$$

Of course we could also use the mean square error or another similar metric, such as the absolute difference, to measure the distance between the curve given by the cumulative layering scheme $\hat{R}(q)$ and the curve given by the TCP rate equation $R(q)$.

Our hill climbing algorithm is now as follows:

Hill Climbing Algorithm:
  Choose $[q_a, q_b]$ and RTT
  Calculate $R(q)$
  Initialize the p_i's $(1 = p_{-1} > p_0 > \ldots > p_{\ell-1} = 0)$
  do
    for $i = 0$ to $l - 2$
      Find the value of p_i $(p_{i-1} > p_i > p_{i+1})$
        s.t. $\hat{R}(q)$ minimizes $NMSE(\hat{R}(q), R(q))$
  while (the p_i's have not converged)

If we define p'_i to be the value of p_i before the execution of the for loop, then we say the p_i's have converged when after the for loop completes we have

$$\sum_{i=0}^{l-2} (p_i - p'_i)^2 < \beta$$

where β is a positive constant. In calculating the p_i's, we chose $\beta = 0.00002$.

We note that in performing the hill-climbing algorithm, we maintain the restriction that the p_i values are decreasing. Similarly, it may be desirable to ensure that the p_i values are somewhat separated, i.e. to ensure that $p_{i-1} - \alpha > p_i > p_{i+1} + \alpha$ for every p_i, where α is the minimum distance between consecutive p_i values. This ensures that it is possible for a layer to receive an increase signal without the layer above also receiving an increase signal. This prevents the situation where subscribers up to layer i find they cannot increase their receive rate because every time they increase their receive rate, so do subscribers to layer $i + 1$, and the combination of increases causes packet loss. In calculating the p_i's, we chose $\alpha = 0.001$.

A priori it is not clear how close we may come to the TCP curve. But experiments over a wide range of values have shown that the hill-climbing algorithm can closely approximate the TCP curve with a cumulative layering scheme.

Example 6: Consider a base layer of $R_0 = 3$ packets per second, i.e., 24 Kbps for 1 KB packets, $\ell = 30$ and a multiplicative layering scheme so the subscription level R_i is equal to $R_0\, c^i$. We set $T = 0.5$ seconds and the nominal round-trip time to be 0.1 seconds. We use the hill climbing algorithm with $[q_a, q_b] = [0.001, 0.1]$, $h = 500$ and the q values linearly spaced. For $c = 1.3$ the explicit (i, R_i, p_i) triples are:

(0, 24.0 Kbps, 0.999)	(15, 1.20 Mbps, 0.312)
(1, 31.2 Kbps, 0.932)	(16, 1.56 Mbps, 0.311)
(2, 40.6 Kbps, 0.931)	(17, 2.03 Mbps, 0.310)
(3, 52.7 Kbps, 0.930)	(18, 2.64 Mbps, 0.248)
(4, 68.5 Kbps, 0.929)	(19, 3.43 Mbps, 0.122)
(5, 89.1 Kbps, 0.928)	(20, 4.45 Mbps, 0.118)
(6, 116 Kbps, 0.927)	(21, 5.79 Mbps, 0.108)
(7, 150 Kbps, 0.926)	(22, 7.53 Mbps, 0.077)
(8, 196 Kbps, 0.925)	(23, 9.79 Mbps, 0.051)
(9, 255 Kbps, 0.924)	(24, 12.7 Mbps, 0.040)
(10, 331 Kbps, 0.923)	(25, 16.5 Mbps, 0.005)
(11, 430 Kbps, 0.922)	(26, 21.5 Mbps, 0.003)
(12, 559 Kbps, 0.921)	(27, 27.9 Mbps, 0.002)
(13, 727 Kbps, 0.920)	(28, 36.3 Mbps, 0.001)
(14, 945 Kbps, 0.901)	(29, 47.2 Mbps, 0.0)

For $c = 1.3$, the graphs of the functions plotting average throughput versus packet loss rate for TCP and FLID with these settings is given in Figure 11. The hill-climbing algorithm provides a better fit than the initial heuristic, particularly at low loss rates.

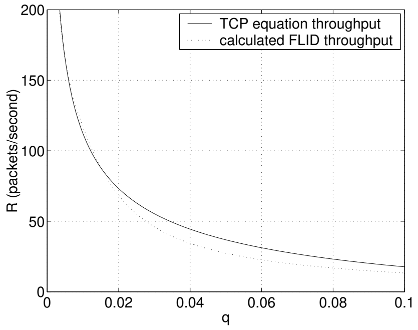


Fig. 11. Comparison of the estimated average throughput as a function of loss probability for FLID and TCP where the p_i values are obtained using the hill-climbing algorithm.

We are conducting further research into determining which p_i settings are appropriate in practice, and whether the $NMSE$ distance metric can be fine-tuned to take into account other considerations that may arise in practice.