On Service Models for Multicast Transmission in Heterogeneous Environments

Matthias Grossglauser
AT&T Labs- Research
180 Park Avenue
Florham Park NJ 07932
mgross@research.att.com

Jean-Chrysostome Bolot
Ensim Corporation
1215 Terra Bella Avenue
Mountain View CA 94043
bolot@ensim.com

Abstract– We examine in this paper the tradeoff between application complexity, network complexity, and network efficiency. We argue that the design of the current Internet reflects a tradeoff between lower network complexity (no state in the network, no signalling) and higher application complexity (rate and error control mechanisms to obtain an adaptive application) assuming a *unicast service model*. For such a service model, a design methodology that leans heavily towards application complexity has proven very successful.

However, we also argue that this tradeoff changes radically for a *multicast/multilayer service model*. There are several reasons for this. First, implementing a multicast/multilayer service requires per-flow state. The *incremental cost* of deploying a slightly more elaborate service model that takes into account flow dependence is much smaller than in the case of unicast. Second, several end-to-end functions, such as channel estimation and error control, are considerably more difficult for multicast/multilayer applications in a large-scale and heterogeneous environment than for unicast applications. Third, the operating point of a pure best-effort network is dictated largely by elastic applications (such as those based on TCP). Unfortunately, this operating point tends to be undesirable for multicast/multilayer applications, as they have for example to use FEC to protect high-priority layers. Other choices similarly lower the network efficiency.

These insights motivate a new service model which slightly departs from the best-effort model, and which trades off a slightly higher network complexity for much lower application complexity and higher network efficiency. We describe this service model and the associated network protocols. The protocol complexity is only marginally higher than that of a simple multicast routing protocol with receiver-initiated join/leave capabilities. The dependencies between multilayer flows are established and maintained as soft state; therefore, no explicit session signalling to establish and tear down flow dependence state is necessary.

I. Introduction

The current Internet provides a simple service, namely a single class best effort service, which relies on simple mechanisms in the network, namely datagram switching and stateless FIFO queueing. The model is attractive because it facilitates the interconnection of networks with different architectures, and it provides flexible resource allocation and good reliability against node and link failure [4]. The underlying network mechanisms are attractive because they are extremely simple and they do not rely on complex control mechanisms or signaling protocols.

However, from a connection's point of view, best effort service amounts in practice to offering a channel with time-varying and hard-to-estimate characteristics such as delay and loss distributions. This makes it hard to predict or guarantee performance such as maximum delay or maximum loss rate to applications. Two approaches to this problem are possible. For clarity of presentation, let us first consider the case of unicast connections.

One approach is to adapt application behavior to the service provided by the network, i.e. to the time-varying characteristics of the channel over which the application data packets are sent. This amounts in practice to incorporating in the application, control mechanisms that attempt to minimize the negative impact of channel characteristics on the quality of the data delivered at the destination. Examples of such mechanisms include playout adjustment (to control delay jitter), rate control (to match bandwidth requirements to available bandwidth), error recovery (to control the impact of packet loss on quality), etc. Clearly, the complexity of applications increases as applications incorporate more such mechanisms.

The other approach is to augment the best effort service with other services providing various degrees of performance guarantees, i.e., providing channels with more predictable (or even fixed) characteristics. It is clear that the complexity of the control mechanisms will decrease as the predictability of the channel characteristics increases. For example, in a network providing constant bit rate (CBR) channels, applications need not include any kind of playout adjustment scheme (since delay jitter is zero) or error recovery scheme (since packets are not lost due to buffer overflow once admitted).

Thus, we observe a clear tradeoff between the complexity of the applications (or rather the complexity of the control mechanisms included in the applications) on one hand, and the complexity of the mechanisms in the network that provide a service with stricter or looser guarantees, i.e. with more or less predictable characteristics.

The balance of the tradeoff discussed above turns out to be somewhat different in the case of unicast and multicast connections. This is because even in the current Internet architecture, the multicast routing protocols do require that some state be kept in routers. Furthermore, this state is updated using protocols that essentially act as signaling protocols. These signaling capabilities *do* exist even in the stateless Internet and are *required* by the multicast routing algorithms. The question then is whether this state information and the underlying signaling capabilities could not be taken advantage of and used to improve the performance of multicast delivery of layered data. *More generally, the* *question is whether a "small" amount of state information in the network can improve overall performance, and if so how small is small enough, and how much of a performance improvement can be expected.*

We discuss this question in this paper. We consider two issues in particular, and we illustrate our points using two kinds of service models and network architectures. The *service models* we examine are 1) the best effort model in which no state is kept in intermediate routers, and 2) a controlled sharing model with renegotiation similar to the RCBR model in [10]. We pick best effort and CBR-like services because they are in a way two extreme points in the space of service models.

The *specific issues* we examine are 1) the tradeoff between complexity at the application level (i.e. at the network edge) versus complexity inside the network, and 2) the impact on performance and architecture complexity of information related to the different flows sent over multiple multicast groups, specifically information about dependencies between flows.

The rest of the paper is organized as follows. In Section II, we explore the design space spanned by network and application complexity. In Section III, we use the insights from Section II to design a new multicast/multilayer service model that trades off a slightly higher network complexity for a vastly lower application complexity. Section IV concludes the paper.

II. Exploring the Design Space

A. Impact of Service Model on Application and Network Complexity

In this section, we study the tradeoff between application and network complexity, by comparing the functionality required in applications and in network nodes for several classes of service models. We identify several core mechanisms both in the application and inside the network. Table I gives an overview of the mechanisms required to use and implement each class of service model. The order of the service model in the table is roughly given by the degree of "predictability" of the service to the application. To the left, best-effort service is completely unpredictable, making no implicit or explicit guarantees at all. To the right, guaranteed rate/constant bit rate (CBR) service guarantees a fixed bandwidth allocated to the application, without any bandwidth sharing through statistical multiplexing.

Our main focus here is not a direct comparison of these service models; rather, we are interested in the shift in relative costs when we assume that the service is used as a multicast/multilayer service. We argue that the *incremental cost* of adding functionality to the network is smaller than in the unicast case, whereas the cost of application functionality is increased.

A.1 Application Complexity

Let us look at application functionality first. **Channel estimation** is necessary whenever an application has no explicit knowledge, through an explicit prior reservation or through an explicit notification from the network, of the end-to-end channel characteristics such as available bandwidth or delay jitter. The channel estimation problem is harder in multicast, as the channel is now a multicast tree shared with other receivers. Furthermore, the use of multiple layers also makes it necessary to estimate the interdependence of these layers. For example, in multicast congestion control schemes that rely on receiver-initiated joins and leaves to/from layers [3], [16], [15], [23], it is not enough to estimate each per-layer channel, but the relative impact of each layer on other layers as well; this is a hard estimation problem, and there is so far little evidence that such approaches are feasible in large groups [9].

Error control encompasses several types of mechanisms to limit the impact of transmission errors such as packet loss. *Error recovery* attempts to "fill in the gap" either by retransmissions (ARQ) or through forward-error correction (FEC). ARQ is often not feasible in delay-constrained real-time communication, and the problem of determining the optimal amount of FEC information is hard [11] (but solvable in specific cases, e.g. [2]) in a single-flow case. However, the joint optimization of FEC on multiple layers is essentially an open problem, and it can be expected that its complexity is much higher, as the dimension of the parameter space increases with the number of layers. *Error resilience* is a system property that minimizes the impact of transmission errors, for example by appropriately mapping the source information onto packets to limit error propagation. This precludes the use of certain coding mechanisms such as inter-frame coding, as these mechanisms are too sensitive to errors; thus, error resilience is bought at the price of coding efficiency [17]. Error resilience is harder to achieve with multilayer transmission, as errors in the base layer can affect other layers as well. For example, if a frame (or a block) of a video transmission is lost in the base layer, then the corresponding information in the enhancement layer becomes useless, thereby making it harder to limit the propagation of the error.

Traffic specification requires that the application be able to specify the expected behavior of the traffic it generates. Traffic specification of multiple layers is more difficult and expensive than specification of a single layer, because of the larger number of parameters to be determined.

A.2 Network Complexity

Let us now look at network functionality. **Buffer management and scheduling** is not fundamentally harder in a multicast context, as they are essentially local mechanisms managing local resources. The same holds for **admission control**, which computes local schedulability of new flows demanding admission[1]. **Signalling** and **per-flow state**, on the other hand, require network-wide conventions for protocols, formats, and policy. For a signalling protocol to be useful, all the nodes in the network have to implement it. Even in the absence of an explicit signalling protocol such as RSVP [24], a node needs to be aware of packet formats and semantics in order to be able to implicitly set up meaningful per-flow state; for example, in order to perform Fair Queueing, nodes must know how they are supposed to classify packets (e.g., source-destination, destination only, type-of-service (TOS) field, etc.) The deployment of signalling and per-flow state is therefore considerably more difficult than the deployment of local mechanisms mentioned above such as scheduling and buffer management. Thus, the major hurdle when deploying a more elaborate network service really

[1] Assuming that the mapping from end-to-end to local QoS is straightforward [8]

Service model		BE	DS-AF	VBR/CL	RCBR	CBR/GL/EF
Application mechanisms	Channel estimation	●	●			
	Error control	●	●	◐(iii)		
	Traffic specification		◐(i)	●		●
Network mechanisms and properties	Buffer management		◐(ii)	●		
	Scheduling		◐(ii)	●		
	Admission control			●	●	●
	Signalling			●	●	●
	Per-flow state			●	●	●
	Peak reservation					●

TABLE I: Five classes of service models (BE=Best Effort; DS-AF=diffserv with Assured Forwarding PHB; VBR/CL=Variable Bit Rate or intserv Controlled Load; RCBR=Renegotiated Constant Bit Rate; CBR/GL/EF=Constant Bit Rate/intserv Guaranteed Load/diffserv with Expedited Forwarding PHB) and their associated application and network functionality. (i) Diffserv requires traffic specification only for traffic aggregates, i.e., for all the traffic of a class at an ingress point [1]; (ii) In the Diffserv framework, scheduling and buffer management act at the granularity of traffic classes, not individual flows; (iii) VBR and the Controlled Load service are designed to admit a nonzero, but very small probability of packet loss; error control is therefore not crucial, but cannot be completely ignored.

is the deployment of signalling and of per-flow state. Their absence represents the major cost[2] advantage of the best-effort service model of the current Internet.

The simplicity of the service model of the Internet, which essentially limited the exchange of control information between nodes to routing updates, was a major factor in its rapid spread. However, multicast delivery requires a signalling protocol such as DVMRP [6] anyway in order to establish, maintain, and tear down multicast trees[3]. In order to correctly forward multicast packets, nodes must know what tree a packet belongs to, and this requires some state information in the nodes. The amount of state might depend on the number of flows, or the number of multicast groups, depending on whether some kind of state aggregation is used. In any case, the point is that some state information is required in network nodes. Therefore, as both signalling and state information are found anyway in a multicast capable network, the incremental cost of using these functionalities to implement a more elaborate service model may be acceptable. This is especially true since we have observed that the use of multicasting and hierarchical coding of multiple layers considerably increases the complexity of application adaptation mechanisms such as error control, channel estimation, and traffic specification.

In summary, in a multicast/multilayer environment, the balance in the application-network complexity tradeoff might well be shifted in favor of a more elaborate service model and more complexity in the network. We argue that this shift is worth being examined carefully.

B. Impact of Service Model on Network Efficiency

B.1 The Notion of Resource Waste

In this section, we formalize in a simple model the main complication arising from multilayer communication. This complication is the *flow dependence* that arises between flows that represent the same information at various qualities. For example, a raw video stream can be compressed into a sequence of information flows [13]. The quality of the decoded signal increases with the number of contiguous flows available to the decoder. However, a flow cannot be used if it is not part of a contiguous subset of flows, and is therefore effectively wasted. We focus in this section on the *resource waste* that arises if the network is not aware of these flow dependencies.

We define resource waste as a resource that is allocated to an entity (packet, flow, task etc.) that does not contribute to the "utility" of the application [20]. The reason for resource waste can be either (i) the utility for the application depends on the availability of multiple resources, all of which must be available to be useful, or (ii) the allocation of a redundant resource (e.g., when transmitting the same information in different representations multiple times, such as with FEC [13] or simulcast of multimedia streams encoded with different quality levels [3]).

Let us focus on (i). We can distinguish two subtypes of resource waste due to dependence.

- **Internal waste:** Internal waste arises when the network does not complete a task, but nevertheless allocates resources to it. These resources are then wasted. Conceptually, internal waste is due to the distributed nature of resource allocation. It could be completely avoided if resource allocation were done in a centralized manner. As an example, consider a packet that is dropped somewhere in the network; the buffer space and bandwidth consumed up to the drop location is wasted. Internal waste could in principle be avoided by the network: the necessary information is available (e.g., credit-based flow control, open-loop deterministic service).
- **External waste:** External waste arises when the network completes a task that is useless to the application due to external conditions. This happens when the task of which the network is aware (e.g. transmitting a packet from application to application) is completed successfully, but with no utility to the application due to these external conditions, such as the dependence between flows. External waste cannot be avoided by the network, as the network (typically) has no knowledge about the external conditions.

The following examples illustrate the concept of external waste: (i) cell vs. packet loss problem in ATM: if the transmission unit

[2] By "cost", we mean not only the cost of development and of equipment, but also the "political" cost of having to establish standards.

[3] We assume *sparse mode multicast* here, i.e., the number of receivers of a typical multicast group is small w.r.t. the overall network size.

of which the network is aware is the cell, then all cells belonging to the same packet as a dropped cell in this packet become useless to the application; (ii) layered coding: the dependencies between hierarchically encoded flows result in some flows being transported without being useful to the application; (iii) FEC: transmitting redundant FEC packets that end up not being used, because no original packets were lost.

External waste can be transformed into internal waste by giving the network additional information about the dependence between resource requests, i.e., by giving the network partial or complete knowledge of the application utility as a function of the resources allocated to an application. For example, by making a cell-switched network aware of packet boundaries, we can avoid the transport of useless cells following the lost one.

Clearly, we would like to minimize the amount of external waste. To do this, we need to examine (i) whether external resource waste occurs easily, (ii) assess its cost when it does occur, and (iii) derive ways to prevent it.

B.2 Sensitivity of External Waste to Packet Loss

Below is a simple thought experiment to illustrate some of the issues arising from flow dependence. Assume that we transmit a (unicast) session through a network. The session consists of n flows $\{F_1, \ldots, F_n\}$ that depend on each other due to hierarchical coding of the source information in the sense that a packet belonging to flow i is only useful to the application if the corresponding packets on flows $1, \ldots, i-1$ have been received as well.

Suppose that the network does not know about flow dependence and that packets are dropped at random with probability p_{loss}, independent of each other, and independent of what flow the packet belongs to. Then the probability that a packet gets through L links successfully is $p_L = (1 - p_{loss})^L$.

We study the following situation: one packet per layer is sent. A packet arriving at the destination is useful to the destination only if all the packets on layers with smaller index have arrived as well. These n packets might represent a hierarchically encoded video frame, for example.

We are interested in the *fraction of useful traffic* arriving at the destination as a function of the per-link probability p_{loss} and the number L of links traversed. Then the expected number of packets making it to the destination is np_L. The number of *useful* packets making it to the destination is $\sum_{i=1}^{n} p_L^i = p_L \frac{1-p_L^n}{1-p_L}$ as the probability that packet i gets through *and* is useful is p_L^i. Thus, the fraction of useless or wasted traffic at the destination is[4]

$$1 - \frac{1-(1-p_{loss})^{nL}}{n[1-(1-p_{loss})^L]} \tag{1}$$

Fig. 1 shows this fraction after $L = 20$ hops, as a function of the per-link loss probability p_{loss} and n, the number of dependent layers. We can see that the fraction of useless traffic u is quite sensitive to both p_{loss} and n: for example, for $n = 10$ layers and $p_{loss} = 1.0e-2$, about 50% of the traffic arriving at the destination is useless. So although about $p_L \approx 82\%$ of the packets reach the destination, only about 40% of the original packets can be used by the application.

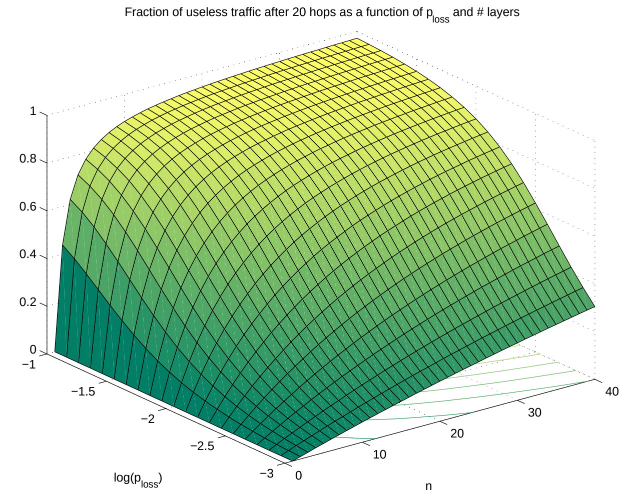

Fig. 1: External resource waste as fraction of total traffic after $L = 20$ hops as a function of the per-link loss probability p_{loss} and the number of layers n.

The underlying problem is that the network assumes a utility function, which it is (hopefully) designed to maximize, considerably different from the utility function of the application. Without knowledge of flow dependence, the network tries to optimize the wrong thing, and performs very suboptimally from the application's point of view in terms of utility w.r.t. to the resources allocated [20].

This difference in the utility functions implies that we really only have two choices: either we operate in a regime where the network and the application utility functions are close, or change the network utility function.

In the present example, the former implies operating in the regime of very small per-link loss probabilities, thereby potentially reducing the achievable statistical multiplexing gain. The latter implies making the network aware of the utility function, and in particular, of the non-additivity of the per-flow utilities. This implies that there is no way around somehow making the network aware of flow dependence.

B.3 Resource Waste and Channel Estimation in the Current Internet

As noted in the introduction, best-effort networks can collapse in the absence of appropriate end-to-end congestion control. Such a collapse manifests itself through high internal waste; most packets do not make it to the destination, thereby using up resources without ultimately being delivered to the application. The current Internet with its best-effort service model has no network-internal congestion control mechanism, such as hop-by-hop flow control, and no mechanism to convey *explicit congestion notification* to applications (we do not count Source Quench as an operational scheme). As a result, end-to-end congestion control mechanisms have to rely solely on the observation of the end-to-end packet transmission process to know

[4] Note that part of this useless traffic is internal waste, and part is external waste: if the network knew about flow dependence, it could drop packets belonging to layers with higher index than the packet lost, thereby avoiding the higher-index packets travel all the way to the destination; however, we would still have internal waste, due to the packets transmission up to the point where they get dropped. This is independent of flow dependence.

about the congestion in the network. In other words, the congestion control mechanism has to *estimate* the characteristics of the end-to-end channel.

We argue that relying on channel estimation constrains the possible operating points at which the network can function. The most prominent channel characteristic that congestion control algorithms are based upon is *packet loss*. For example, TCP views a packet loss as an indication that the network is overloaded and reacts by reducing the size of its congestion window [12]. Although TCP is not based on an a-priori specified target operating point from which a control algorithm is derived, which makes it difficult to characterize that operating point, simulation studies [21] and analytical models [14] as well as evidence from Internet measurements [18] clearly show that TCP's operating point is at rather high packet loss rates ($> 1\%$). We claim that this is not an artifact of TCP, but instead a direct consequence of congestion control based on loss estimation.

Suppose we want to devise an end-to-end congestion control algorithm similar to that of TCP which keeps the network at a much lower packet loss rate (say, 10^{-5}). How would such an algorithm be designed? Assume that at any point in time, packets are metered out into the network at a certain rate. The congestion control algorithm then has to decide if this rate is too high or too low in order to continually adjust the rate. However, in order to make this decision with a certain confidence, the controller must be able to *observe* enough samples of the relevant channel property (here, the bottleneck bandwidth) in order to avoid making wrong decisions too often. But the number of samples required for this depends in turn on the operating point, in the sense that many fewer samples are required to reliably estimate a high loss rate than a low loss rate. For example, a TCP-like algorithm would have to observe on the order of 10^5 packets before being able to reliably decide that the current loss rate is probably lower than the desired loss rate of 10^{-5}, and that it is therefore safe to increase the packet rate. It is clear that this is not feasible in practice, as such a controller would be prohibitively slow to react to changes in network conditions.

The reason why an operating point where the packet loss rate is quite high is not problematic in TCP is that TCP only incurs internal resource waste. When a packet is missing from a TCP session, it is simply retransmitted, and the gap is eventually filled. Thus, the cost of a lost packet is roughly equal to the cost of transmitting an additional packet. However, external resource waste is much more sensitive to the loss rate in the presence of flow dependencies and when retransmission is not an option, such as with real-time video and audio communication. The example in the previous section shows that the implicit operating point of TCP would incur unacceptably high external resource waste[5]. Other control mechanisms competing with TCP for network resources then have no choice but to operate at the same operating point; if they attempted to operate at a lower loss rate, they would be starved by TCP. Thus, competition with TCP effectively precludes applications from operating at a loss rate where external resource waste is low. This is why less aggressive versions of TCP might be greeted with interest, and would operate at a point with less external resource waste, but they are typically not used in practice.

[5]Note that the same operating point would be reached with so-called TCP friendly UDP-based applications.

B.4 FEC to Hide Resource Waste

One way of getting around the above problem of excessive external resource waste at TCP's operating point is to use forward error correction (FEC) techniques to hide packet loss from the application. FEC achieves this by adding redundancy information to each block of source information. The original message can then be reconstructed even in the presence of loss.

The motivation for an application to use FEC is clear if one assumes that the application attempts to use a given *fixed* channel in an optimal way. However, suppose all applications sharing the network use FEC; this increases the total load of the network, and may actually worsen the channel characteristics as seen by each application. Therefore, it is not evident if the *generalized* use of FEC is beneficial or detrimental.

In the appendix, we discuss this problem using a simple model. We make the following observations. If an application views the channel as *fixed*, i.e., the packet loss probability is independent of its own packet generation rate, then increasing the packet generation rate obviously appears beneficial to that application. However, if all (or most of) the applications use FEC, then it is possible that additional redundancy is actually detrimental. In our model, this occurs when the total traffic load exceeds a certain threshold. In this case, the capacity of the network (the "goodput") can be increased by *decreasing* the traffic load and operating at a smaller loss rate, requiring *less* redundancy. Mechanisms that allow such an operating point of small loss probability and little redundancy result in more optimal use of network resources. Therefore, the overall benefit of the generalized use of FEC in a network needs careful examination.

C. Summary

We have examined a number of service models, and we have seen that to each model corresponds a balance between application complexity, network complexity, and network efficiency. Considering in particular the best effort service model provided by the current Internet, we find that for multilayer multicast applications,

- **Application complexity** is high: Channel estimation is hard because the channel is a multicast tree shared with other receivers; thus it is necessary to estimate the relative impact of each layer on other layers, leading to very hard joint estimation procedures. Error control and resilience raises thorny issue of joint FEC optimization across several interdependent layers.
- **Network complexity** is very low in the case of unicast transmission (no state in nodes, no signalling), however it is much higher (though much of it hidden) in the case of multicast transmission. Indeed, multicast delivery requires some sort of signalling protocol to establish, maintain, and tear down multicast trees. Furthermore, some amount of state (per-flow, or per-group) must be maintained in nodes.
- **Network efficiency** is limited by constraints by the network model, which forces applications to rely on channel observation. The flow dependencies in multilayer transmission require a network operating point where the loss rate is very low. However, this makes channel estimation through packet loss difficult.

Furthermore, the competition with TCP means that the operating point is determined by TCP's behavior, which keeps loss rates high. Finally, the generalized use of FEC to decouple the channel as seen by the congestion control mechanism from the channel used by the application is problematic.

Thus, in a multicast/multilayer environment, the network complexity which is required anyway does not translate into low application complexity. In the next section, we illustrate a departure from the best-effort service model, and describe a service model which trades off a slightly higher network complexity for much lower application complexity and higher network efficiency.

III. A Renegotiation-Based Service Model for Layered Multicast

The goal of this section is to illustrate our insight about the tradeoff between network and application complexity by generalizing the RCBR service model to a multicast setting. We picked RCBR as our starting service model because it is almost the other extreme, in the service space, to the best effort model examined earlier in the paper. We demonstrate how the Generalized RCBR (GRCBR) protocol, which requires little more network complexity in terms of state and processing than a simple multicast routing protocol, considerably simplifies the design of multicast/multilayer applications and avoids external resource waste by taking flow dependence into account.

A. Definition of the GRCBR Service Model

The Renegotiated Constant Bit Rate (RCBR) service is a simple service which augments a constant bit rate (CBR) service with a renegotiation mechanism that allows a source to request a change to the guaranteed rate [10]. An application relying on the RCBR service retains many of the advantages it would get with a CBR service, namely a very low loss probability and no queueing delay. Inside the network, RCBR requires only small buffers and no complex scheduling disciplines. Furthermore, it has been shown that RCBR can extract most of the statistical multiplexing gain of traffic with bandwidth fluctuations on multiple time-scales.

Renegotiation Semantics. Let us then first examine the semantics of renegotiation in GRCBR. Recall that in RCBR renegotiation succeeds only if all the links on the path as well as the receiver itself can accommodate the new requested bandwidth. We therefore extend this model to a multicast tree simply by requiring that for a renegotiation to succeed in GRCBR, all links and receivers making up the multicast tree can accommodate the new bandwidth. We call this type of renegotiation a *closed renegotiation* (CRNG).

We now discuss the necessity for a second type of renegotiation called *open renegotiation* (ORNG). Actually, one of the goals of using multiple hierarchical layers is precisely to use this spare bandwidth as efficiently as possible. One way to achieve this is to limit the "fate sharing" in the multicast tree. An open renegotiation achieves this as follows: When an open renegotiation request arrives on a link, the link attempts to reserve the requested bandwidth; if it cannot, then the link and the whole subtree fed by this link is renegotiated to zero bandwidth and effectively disconnected. The flow may be reinstated on this link upon its next (open) renegotiation, if spare bandwidth has become available in the meantime.

The rationale for the two types of renegotiation are as follows. A receiver is assumed to require a minimum set of layers in order to satisfy the minimum quality requirements of the application. The receiver cannot accept losing these layers even temporarily; only closed renegotiations are therefore acceptable between the source and this receiver. The receiver might also want to receive lower-priority layers whenever possible. This means that the corresponding flow could grab unused resources when available, and thus maximize network utilization and receiver utility. For such a flow, an open renegotiation between the sender and the receiver would be acceptable.

Admission Control. The role of admission control is to ensure that a flow is only admitted on a link if the renegotiation failure probability after admission of the new flow remains small for the new flow and for all previously admitted and still active flows. Therefore, we propose the following design rule: If a flow has been admitted on a link, then both open and closed renegotiation can be performed on this link; insufficient unused bandwidth on this link can therefore cause a failure of a CRNG for this flow; If a flow has *not* been admitted on a link, then only open renegotiations (ORNG) can be performed on this link. Insufficient unused bandwidth can therefore affect only receivers *downstream* from this link.

This rule partitions the multicast tree into a guaranteed (G) tree rooted at the sender[6], and a set of non-guaranteed (NG) subtrees hanging off the G-tree (cf. Fig. 2). A CRNG issued by the sender is changed into an ORNG when the request traverses from the G-tree into one of the NG-subtrees.

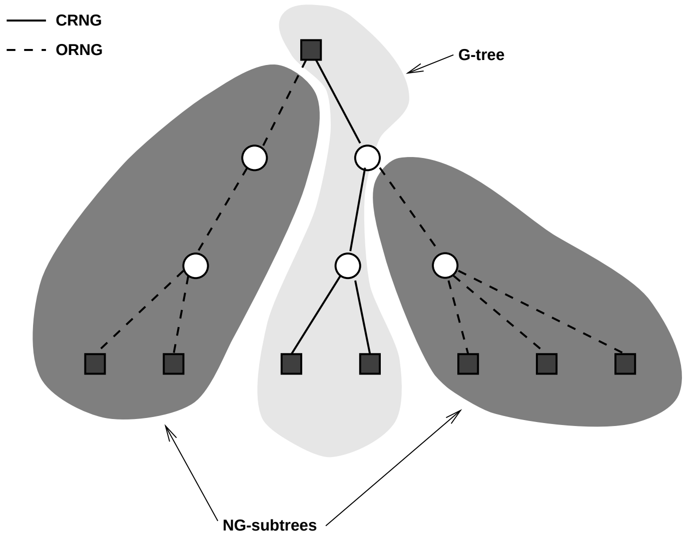

Fig. 2: The multicast tree consists of an admitted or guaranteed part (G-tree) and a "renegotiated best-effort" part (set of NG-subtrees).

Atomic Processing of Several Renegotiation Requests. In Section II-B.1, we discussed the advantage of making the network aware of flow dependencies to avoid external resource waste. Let us now examine how to exploit a knowledge of flow

[6]We also require that if a flow has not been admitted on a link, then it will not be admitted on any link downstream from it.

dependence in the context of the GRCBR protocol. Consider the case of an application sending hierarchically encoded data over n flows. We would like a renegotiation request for layer i to be satisfied only if those for layers $1, \ldots, i-1$ are satisfied as well. More generally, if the available bandwidth on this link is insufficient to accommodate all of the n requests, they should be satisfied in the order desired by the application. We see then that the only functionality we have to add to the renegotiation protocol is the ability to atomically process a sequence of renegotiation requests. The semantics of such an atomic renegotiation are to atomically free the bandwidth held by the respective flows, and then request the new bandwidth in sequential order (cf. Fig. 3).

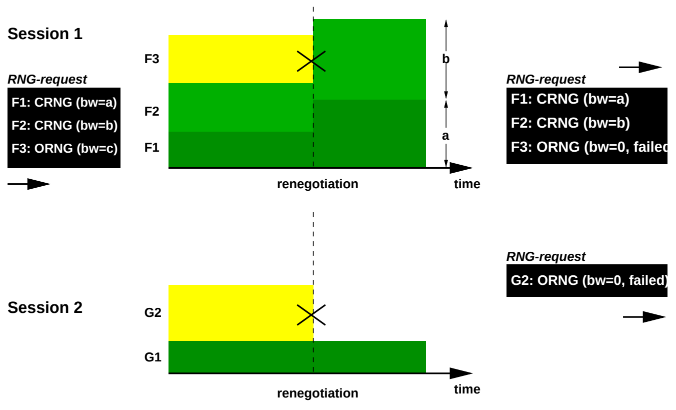

Fig. 3: An atomic renegotiation request arrives for a multilayer session (F_1, F_2, F_3). The requests for more bandwidth for flow F_1 and F_2 can be granted, but at the expense of blocking the request for F_3, and of preempting another non-guaranteed flow G_2 of a different session. The renegotiation request for $F_{1,2}$ is passed on downstream, along with the request to release the bandwidth held by F_3. A new message is also generated to release the bandwidth held by G_2.

B. Implementation Issues

We discuss implementation issues for the GRCBR service model. We focus mainly on signalling and network state required to implement this service model.

Preemption of NG Flows. Admission control limits the number of guaranteed flows on a link to keep the renegotiation failure probability at some small value. The non-guaranteed flows are in a sense "renegotiated best-effort" flows; the renegotiation failure probability of such flows may be high. Therefore, our service model requires preemptability of non-guaranteed flows in order to respect the priority of guaranteed flows. Admission control of the guaranteed flows assumes that they can potentially use up all of the available bandwidth[7]. Therefore, when insufficient unused bandwidth is available for an (open or closed) renegotiation request for a guaranteed flow, then this bandwidth should be obtained by preempting non-guaranteed flows, if any.

Flow Dependence State for NG Flows. While the selection criterion of which NG-flow(s) to preempt depends on local policy (such as fixed priority or smart market bidding [22]), it is clear that preempting NG-flows without taking flow dependencies into account may again lead to external resource waste. Since the preemption of NG-flows happens asynchronously, it is necessary to introduce a *flow dependence state* for NG-flows to avoid external resource waste due to preemption. This state can take the form of two pointers high-prior and low-prior for an NG-flow that points to the two flows of the same multilayer session with the next lower and the next higher priority, respectively. Fortunately, setting up this state does not require an additional session control signalling message, because it is implicitly conveyed by the bundled renegotiation request. As the order of the flows in the bundled renegotiation request implicitly contains the flow dependence information, the pointers can be set when a bundled renegotiation request is processed by a node. The NG-flow's dependence information is therefore maintained as *soft state* in the sense there is no need for an additional *session management protocol* to convey flow dependence information from applications to the network. When one or several NG-flows are preempted on a link, then the bandwidth they hold on the subtree fed by this link should be released.

No Network-Initiated Reinstatement of Preempted Flows. We do not include a mechanism for network-initiated reinstatement of preempted NG-flows. The reason is that in a cyclic network such as a general mesh network, there can be cycles in the resource dependency graph. Without additional mechanisms such as imposing a total order on flows or centralized control of preemption/reinstatement, the cyclic resource dependency that may arise among flows sharing multiple resources in the network, *thrashing* may occur. Thrashing can dramatically increase the load of the signalling system, and decrease network throughput, as the flows involved in the thrashing are not in a coherent state most of the time[8]. Note that multicast may exacerbate the likelihood of such cycles occurring, as a preemption or reinstatement of a flow can be "seen" by a potentially large subtree. We break these resource dependence cycles by only allowing the network to preempt flows, but not to reinstate them. Reinstatement of a flow can only occur upon its next renegotiation. As renegotiations are issued by the source, and are therefore independent of preemption inside the network, resource dependency cycles cannot arise.

Signalling for GRCBR. The signalling protocol for RCBR is a simple two-phase reservation protocol. The first phase makes a temporary reservation on all the links it traverses in the forward direction (from the source towards the receivers). The second phase in the reverse direction (towards the source) either commits or cancels this temporary reservation. We call the signalling message in the first phase *renegotiation request* (RNG-request), and in the second phase *renegotiation response* (RNG-response).

GRCBR requires a two-phase reservation protocol similar to its unicast counterpart. However, because of the different semantics of open and closed renegotiation, the paths that signalling messages would take in the case of a renegotiation failure for both types of messages may be different. With closed renegotiation (CRNG), all the links in the G-tree must be able to accommodate the new bandwidth for the renegotiation to suc-

[7]Although it is conceivable that a service provider would set the available bandwidth for guaranteed flows to less than the link capacity, in order to set aside some minimal bandwidth for non-guaranteed (renegotiated best-effort) flows.

[8]A flow is in a coherent state if no internal waste arises from information flowing over links without reaching receivers downstream from this link.

ceed; therefore, a RNG-request fails and is terminated when it encounters a link that cannot accommodate the requested bandwidth. In that case, a negative renegotiation response is sent back up the tree to cancel the tentative reservation. It is also necessary to undo tentative reservations in the subtrees hanging off the path leading from the source to the failed link. For this, an additional third signalling message of type CRNG-undo is used.

With open renegotiation (ORNG), when a link has insufficient bandwidth to accept a renegotiation request, the subtree fed by this link is temporarily cut off. Of course, it may later be reinstated by the next renegotiation for this flow. The renegotiation request upon encountering such a link has to travel further down the tree, in order to release the bandwidth reservation held by this flow in the blocked subtree. For example, in Figure 4, if an ORNG for a flow on link DE fails, then forwarding for that flow on link DE is suspended. However, the bandwidth reservation on ER_5 and ER_6 must be canceled. Also, in this case, a negative RNG-response should be sent upstream to release reservations that have become useless as a result (in the previous example, the link SD).

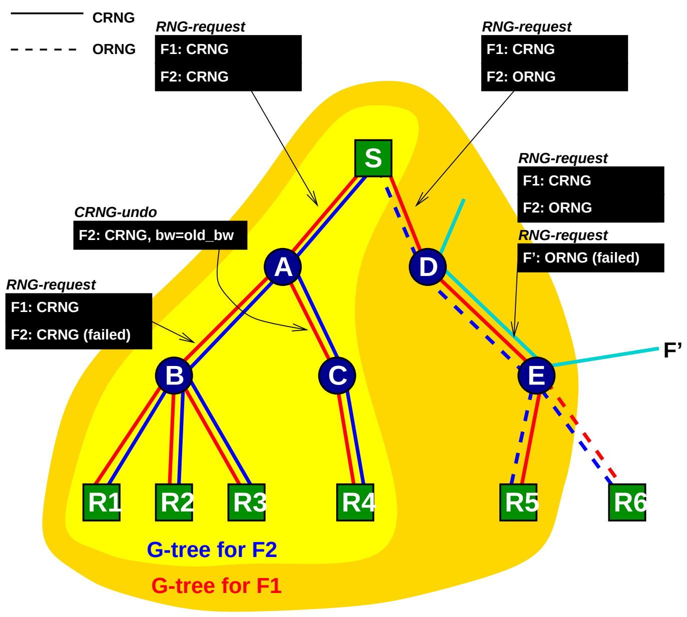

Fig. 4: An example tree with two flows F_1 and F_2 emitted by source S, where F_1 has higher priority than F_2. Four receivers $R_1, \ldots, R_4$ receive both F_1 and F_2 as guaranteed flows; R_5 only receives the base layer F_1 as guaranteed (G), but the enhancement layer (F_2) as "renegotiated best effort" (NG) - it may be interrupted during congestion; R_6 receives both flows as NG (note that flow priority for R_6 is nevertheless ensured: R_6 will never receive F_2 without also receiving F_1.) A failed CRNG on link AB with corresponding CRNG-undo on link AC is shown, as well as the transformation of a CRNG into a ORNG at the boundary between the G-tree and the NG-subtree for F_2, and finally the preemption of flow F' by the RNG-request for (F_1, F_2) on link DE (F' is a NG flow of another session.)

Unfortunately, CRNG and ORNG requests cannot follow different paths if we are to bundle renegotiations for multiple flows and process them atomically on each link in order to respect their dependencies. If a source initiates a bundled renegotiation request, possibly combining ORNG and CRNG (some of which may also be changed into ORNG if the corresponding flow has NG-subtrees), then the processing of this request must remain atomic throughout the tree in order to respect the priorities of the flows. Therefore, the bundled renegotiation must follow a common path that permits the correct processing of both CRNG and ORNG.

This can be achieved as follows. The renegotiation request spans the entire tree from the source to all the receivers. A renegotiation response is generated only when the request reaches a receiver. The responses are merged such that there is only a single (bundled) response on each link.

When an NG-flow is preempted as a result of a CRNG-request by another flow for which sufficient bandwidth is not available, then the situation is as if an ORNG-request for this NG-flow had failed on this link (flow F' on link DE in Fig. 4): an ORNG-request with a set failed-flag is generated for this flow. Note that the high-prior and low-prior pointers contain the flow dependence information and allow to avoid external resource waste upon preemption of a flow. In fact, suppose a flow F' is a candidate for preemption. If flow F''s low-prior points to F'', then F' should be preempted only if F'' is preempted as well.

Let us summarize the GRCBR scheme. The service model offers a renegotiable CBR flow from a source to a set of receivers. Some receivers can be connected to a GRCBR multicast tree only through one or several "renegotiable best-effort" (NG) links. These receivers can experience interruption in the reception of such a flow. If a receiver is connected directly to the G-tree, then it will not see interruption of the flow if the source only issues CRNGs.

The data path of the network is as simple as in RCBR. Network buffering is minimal, and no sophisticated scheduling algorithms are necessary. The signalling path essentially receives bundled renegotiation messages from an upstream link, forwards them to the appropriate downstream links, and awaits the responses. It merges the responses and passes them upstream. The CRNG-undo message is only necessary when CRNG fails, which is a rare event. Non-guaranteed flows can be preempted on a link to make bandwidth available for guaranteed flows.

We do not discuss flow setup and teardown in detail here, as it is essentially an orthogonal issue. Let us just remark that flow setup and teardown (e.g., receiver-initiated join and leave requests) should be bundled in the same way as the corresponding renegotiations, in order to ensure that (i) the flows belonging to a given session follow the same path in the network, and (ii) flow dependence is respected in admission control (layer i is guaranteed only if layers $1, \ldots, i-1$ are guaranteed as well). Furthermore, let us remark that flow join/leave will typically occur much less frequently than in schemes that rely on receiver-initiated join/leave as a congestion control mechanism, such as those proposed in [3], [16], [15], as flow preemption and reinstatement happen implicitly in GRCBR. This is important as join/leave operations are quite slow and costly.

C. Summary

Let us summarize the GRCBR multicast service model. A source sends a number of flows to a set of receivers. By grouping renegotiations for these flows in the order of their relative priorities, the GRCBR service guarantees to transport a contiguous set of flows to the receivers, such that no external waste arises. For each flow, a receiver may be connected directly to the G-tree, or it may be connected to an NG-subtree. If the receiver is connected to the G-tree of a flow, then this flow is never disrupted, provided the source only issues CRNGs for this flow; if the receiver is connected to the NG-subtree, then it may experience disruption of the flow, depending on network congestion.

More generally, we have described a service model that trades off a slightly higher network complexity for lower application complexity and higher network efficiency. We briefly compare the GRCBR service model to a pure best-effort service model described in Section II.

- **Application complexity** is dramatically reduced under the GRCBR service model. First, *channel estimation* is essentially unnecessary, as the channel is a simple CBR pipe with very low packet loss rate and small delay and delay jitter. Therefore, there is no need to estimate round-trip delays, packet loss rates etc. [9] *Error control* is also simplified, as a flow either reaches a receiver with very low packet loss probability, or not at all (for NG flows). The only error control necessary is for the source to be able to adapt to rare failures of closed renegotiations and adapt its sending rate until the next successful renegotiation. *Traffic specification* is necessary for guaranteed flows in order to perform admission control. However, measurement-based admission control (MBAC) may be used to relieve the application of the burden of traffic specification.
- **Network complexity:** the GRCBR service model relies on a very simple data path, and could be implemented on top of an intserv Guaranteed Rate (GR), diffserv Expedited Forwarding (EF), or ATM CBR service. Renegotiation messages require limited processing in a node (the common case is a simple test for each layer if the requested amount of bandwidth is available, and potentially the preemption of a NG flow). The implementation could make use of RSVP resv and path messages, for example. GRCBR does require per-flow network state, of course. As the multicast service already uses per-flow state anyway, the incremental cost of including the GRCBR state is small.
- **Network efficiency:** as GRCBR processes renegotiation requests for several layers atomically, external waste can be completely avoided. We have argued earlier that if the network is unaware of flow dependencies, we can avoid external resource waste only by either keeping the loss probability very low (which reduces link utilization and is problematic when sharing bandwidth with TCP flows) or by protecting high-priority layers with FEC (which wastes bandwidth by transmitting redundant information, and can actually reduce the overall network capacity). Simply allowing for atomic processing of renegotiations eliminates external resource waste.

IV. Conclusion

The main issue of this paper is the tradeoff between application complexity, network complexity, and network efficiency. We have argued that the design of the current Internet reflects this tradeoff assuming a *unicast service model*. For such a service model, a design methodology that leans heavily towards application complexity (sometimes referred to as the "end-to-end argument" [19]) has proven very successful.

We have also argued that this tradeoff changes radically for a *multicast/multilayer service model*. There are several reasons for this. First, implementing a multicast/multilayer service requires per-flow state. The *incremental cost* of deploying a slightly more elaborate service model that takes into account flow dependence is much smaller than in the case of unicast. Second, several end-to-end functions, such as channel estimation and error control, are considerably more difficult for multicast/multilayer applications that have to function in a large-scale and heterogeneous environment than for unicast applications. Third, the operating point of a pure best-effort network is dictated largely by elastic applications (such as those based on TCP). Unfortunately, this operaing point tends to be highly undesirable for multicast/multilayer applications, as they face two undesirable choices: either accept high external waste, or use FEC to protect high-priority layers. Both choices lower the network efficiency.

We have substantiated this claim by discussing the GRCBR service model for layered multicast, which relies on a renegotiation mechanism for the controlled sharing of bandwidth. GRCBR inherits from RCBR the simplicity of the data path, requiring only minimal buffering and FIFO scheduling. External resource waste is avoided by respecting flow dependencies. We have argued that application complexity is greatly reduced, as channel estimation and error control become very simple. The price for this is a small amount of additional per-flow state in the network (in addition to the state necessary for the forwarding tree), and the ability to process several renegotiation requests atomically.

Acknowledgments

The authors would like to acknowledge very helpful comments from Suhas Diggavi.

[9] A receiver might choose to estimate the renegotiation blocking probability for NG flows over a longer time-scale, in order to subscribe to or unsubscribe from layers. However, if the receiver is too aggressive and joins a group which will be blocked most of the time, this does not degrade the higher-priority layers, because the network is aware of flow dependencies.

References

[1] Y. Bernet, J. Binder, S. Blake, M. Carlson, B. E. Carpenter, S. Keshav, E. David, B. Ohlman, D. Verma, Z. Whang, and W. Weiss. A Framework for Differentiated Services. *Internet Draft (work in progress)*, February 1999.

[2] J-C. Bolot, S. Fosse Paris, and D. Towsley. Optimal joint rate/FEC control for Internet telephony. In *IEEE Infocom '99*, New York, NY, March 1999.

[3] S. Y. Cheung, M. Ammar, and X. Li. On the Use of Destination Set Grouping to Improve Fairness in Multicast Video Distribution. In *IEEE INFOCOM '96*, San Francisco, Calif., USA, March 1996.

[4] D. D. Clark. The design philosophy of the DARPA Internet protocols. In *Proc. ACM SIGCOMM '88*, pages 106–114, Stanford, CA, August 1988.

[5] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[6] S. E. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proc. ACM SIGCOMM '88*, Stanford, Calif., August 1988.

[7] Suhas Diggavi and Matthias Grossglauser. Information Transmission over a Finite-Buffer Channel. Submitted for publication, September 1999.

[8] V. Firoiu and D. Towsley. Call Admission and Resource Reservation for Multicast Sessions. In *Proc. IEEE INFOCOM '96*, San Francisco, California, March 1996.

[9] R. Gopalakrishnan, J. Griffioen, G. Hjalmtysson, and C. Sreenan. Stability and Fairness in Layered Multicast. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '99)*, Basking Ridge, New Jersey, USA, June 1999.

[10] M. Grossglauser, S. Keshav, and D. Tse. RCBR: A Simple and Efficient

Service for Multiple Time-Scale Traffic. *IEEE/ACM Transactions on Networking*, 5(6):741–755, December 1997.

[11] B. Hochwald and K. Zeger. Tradeoff Between Source and Channel Coding. *IEEE Trans. on Information Theory*, 43(5), September 1997.

[12] V. Jacobson. Congestion avoidance and control. In *Proc. ACM SIGCOMM '88*, pages 314–329, Stanford, Calif., USA, August 1988.

[13] G. Karlsson and M. Vetterli. Packet Video and Its Integration into the Network Architecture. *IEEE Journal on Selected Areas of Communications*, 7(5), June 1989.

[14] T. V. Lakshman and U. Madhow. Performance Analysis of Window-based Flow Control using TCP/IP: Effect of High Bandwidth-Delay Product and Random Loss. In *Proc. 5th IFIP Conference on High Performance Networking (HPN) '94*, Grenoble, France, June 1994.

[15] X. Li, S. Paul, and M. Ammar. Layered Video Multicast with Retransmissions (LVMR): Evaluation of Hierarchical Rate Control. In *Proc. IEEE INFOCOM '98*, San Francisco, Calif., USA, March 1998.

[16] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proc. ACM SIGCOMM'96*, pages 117–130, Stanford, CA, Sept. 1996.

[17] S. McCanne, M. Vetterli, and V. Jacobson. Low-complexity Video Coding for Receiver-driven Layered Multicast. *IEEE Journal on Selected Areas of Communications*, 16(6):983–1001, August 1997.

[18] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. ACM SIGCOMM '97*, Cannes, France, September 1997.

[19] J. Saltzer, D. P. Reed, and D. D. Clark. End to end arguments in system design. *ACM Transactions on Computer Systems*, pages 277–288, Nov. 1984.

[20] S. Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal on Selected Areas of Communications*, 13(7), 1995.

[21] S. Shenker and L. Zhang. Some Observations on the Dynamics of a Congestion Control Algorithm. *ACM SIGCOMM Computer Communication Review*, 20(5), October 1990.

[22] F. Toutain and O. Huber. A General Preemption-Based Admission Policy Using a Smart Market Approach. In *Proc. IEEE INFOCOM '97*, Kobe, Japan, April 1997.

[23] Lorenzo Vicisano, Luigi Rizzo, and Jon Crowcroft. TCP-like Congestion Control for Layered Multicast Data Transfer. In *Proc. IEEE INFOCOM '98*, San Francisco, Calif., USA, March 1998.

[24] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, September 1993.

Appendix: What if Everybody Uses FEC?

We consider the following simple model. A source transmits a message through a network to a receiver. The message is encoded into packets. We model the network as an erasure channel, which either perfectly transmits a packet, or completely loses a packet. Under mild assumptions on the erasure process, such a channel can be shown to have Shannon capacity $C = (1-p)L\lambda$ bits per second, where λ denotes the packet arrival rate and where L is the packet size in bits (assumed constant) [7]. The Shannon capacity expresses the amount of information per time unit that can be transmitted reliably over a channel [5]. We consider the capacity as a function of λ.

First, let us consider the ideal choice of λ if p is fixed, which can be viewed as finding the packet generation rate for a single flow in a system that is shared among a large number of flows, such that *the loss probability is not affected by this individual flow's bandwidth*. As $(1-p)L\lambda$ increases monotonically with λ, a flow can obviously increase its capacity by increasing its packet arrival rate.

Second, let us consider the ideal choice of λ if p depends on λ, which can be viewed as finding the packet generation rate for a single flow if all flows sharing the system behave in the same way, i.e., they generate packets at the same rate λ and use the same channel coding. We assume that $p(\lambda)$ is increasing. If $p(\lambda)$ converges to one quickly enough for $\lambda \to \infty$, there exists a finite λ^* such that the capacity is maximized. Assuming $p(\lambda)$ is differentiable, this λ^* satisfies the condition

$$\lambda^* \frac{d}{d\lambda^*} p(\lambda^*) + p(\lambda^*) - 1 = 0. \tag{2}$$

If $p(\lambda)$ is convex-U, then λ^* is upper-bounded by $\tilde{\lambda}$ which is the unique solution of

$$\tilde{\lambda} \frac{d}{d\tilde{\lambda}} p(\tilde{\lambda}) = 1. \tag{3}$$

Whenever $\lambda < \lambda^*$, the system is in a regime where all flows can increase their capacity by increasing λ. The packet loss rate does increase with λ, but it can be compensated for through a modest increase in redundancy in the encoding. However, when $\lambda > \lambda^*$, increasing λ will actually decrease the capacity: the packet loss probability, and therefore the redundancy necessary to compensate for it, increases too quickly.

As an illustration, we can approximate the packet loss probability p with the buffer overflow probability of an $M/M/1$-queue with buffer size B and service rate μ. Suppose the queue is shared by k statistically identical flows. Let $\rho = k\lambda/\mu$ denote the normalized load. The overflow probability of this queue is given by[10]

$$p = \rho^{B+1}. \tag{4}$$

Equation (3) becomes

$$\begin{aligned} \tilde{\lambda}(B+1)\left(\frac{k\tilde{\lambda}}{\mu}\right)^B \frac{k}{\mu} &= (B+1)p(\tilde{\lambda}) = 1 \\ p(\tilde{\lambda}) &= \frac{1}{B+1}. \end{aligned} \tag{5}$$

Therefore, if the packet loss probability $p(\lambda) \geq 1/(B+1)$, we are in the regime $\lambda > \lambda^*$ where the capacity *decreases* with the packet generation rate λ. The capacity is suboptimal, because too much rendundancy is necessary to compensate for loss. To increase the capacity, the flows would have to *lower* their packet generation rate λ and use *less* redundancy. Note that $1/(B+1)$ is a packet loss probability frequently exceeded in Internet routers (B is typically on the order of several hundred packets in current backbone routers), suggesting that it is questionable if the universal deployment of FEC applications would be beneficial in the current Internet.

[10] Note that the packet loss rate of a finite buffer queue of size B is upper-bounded by the probability of exceeding the threshold B in an infinite-buffer queue.