Congestion Control for Best-Effort Service: Why We Need a New Paradigm

The current congestion control paradigm assumes that end users will use a single mandated algorithm. While the work done in this area has proven to be of great value, we need to recognize as a community that this paradigm is clearly inappropriate for future public networks.

Christopher Lefelhocz, Fore Systems
Bryan Lyles, Xerox Palo Alto Research Center
Scott Shenker, Xerox Palo Alto Research Center
and Lixia Zhang, UCLA

The first generation of telecommunication networks were circuit-switched. In such networks, each circuit is allocated resources along a path for its exclusive use; there is no uncertainty about the bandwidth or delays along the path. Packet switching introduced a very different mode of communication. In most packet-switched networks,[1] sources send their data packets into the network without any prereserved resources, and the network exerts its "best effort" to service the packets. The advantage of packet switching is that it allows the network resources (bandwidth, buffers) to be statistically shared among all sources. This is especially important for computer communications, since data traffic tends to be rather unpredictable and bursty; prereserving resources would lead to low utilization levels, whereas the statistical multiplexing of packet switching allows one to achieve much higher utilization levels. The quality of service (in terms of packet delivery delays and drops due to buffer overflow) of best-effort service depends not only on the network actions (which the network can control), but also on the offered load (which the network cannot control). Thus, in best-effort service, the network tries to forward all packets as soon as possible, but cannot make any quantitative assurances about the quality of service delivered.

The unpredictable and bursty nature of computer traffic not only prevents the network from making quality assurances, but also creates the problem of congestion. The ratio of average demand to peak demand on any particular link is quite high. Thus, while it is advisable to provision the network to have adequate resources to satisfy the average demand, it is not economically feasible to provision the network to satisfy the peak demands. Consequently, there will be times when the network is momentarily overloaded when many sources happen to send their data simultaneously. To control this congestion, the network must provide feedback to the users indicating that the network is currently, or is about to become, overloaded; in response to such a congestion signal, the users should inject their packets into the network more slowly. These processes, the feedback from the network and the source response, form the fundamental basis of congestion control.[2] These congestion control algorithms have been the subject of intense study, and the community has made tremendous advances, both in developing effective algorithms and in understanding how they work; see [2–13] and references therein for a few representative references.

Given the bountiful fruits of this research endeavor and the obvious success of the Internet with its congestion control algorithms, why are we revisiting this problem? To understand this, note that research on congestion control has almost exclusively focused on the nature of the congestion signals and of the response to those signals. The current congestion control paradigm is based on the tacit assumption that we, as a community, can design the response of end users to congestion signals. In short, this paradigm assumes that all users are willing and able to cooperate. These congestion control designs do not work if some users choose to misbehave; in particular, greedy users can capture more than their share of bandwidth by not responding to congestion signals. Such greedy users not only capture more bandwidth for themselves, but also seriously degrade the service obtained by cooperating users. Thus, the current paradigm provides reasonable service to users only if all (relevant) users cooperate.[3]

The assumption of user cooperation has been, for the most part, valid in the Internet. Until recently, the Internet user community was a small, relatively close-knit, and technically knowledgeable community. There has been widespread adherence to informal rules of etiquette (not just for congestion control, but also for the proper use of electronic mail and other issues). Moreover, the widespread deployment of UNIX and its variants as the operating system of choice allowed the "standard" congestion control algorithms, transmission control protocol (TCP) Slow-Start [6], to be almost universally deployed.

[1] *X.25 and its descendents are exceptions.*

[2] *As observed by Jain [1], congestion control really has two separate components: congestion avoidance and congestion recovery. We will be focusing on congestion avoidance in this article.*

[3] *We say relevant users since the service obtained by a user in one portion of the Internet typically does not depend in detail on the behavior of another user far away.*

We are now facing a rather dramatic shift in the nature of wide-area computer networks. First, the Internet is now publicly accessible, and the user community is no longer small nor close-knit. We cannot expect adherence to informal social rules to continue to be the norm. In a public network, general appeals to a vague "public interest" will not be sufficient to induce cooperation. Moreover, the user population, or more correctly the set of host machines, is much more heterogeneous, so deployment of congestion control algorithms through the distribution of UNIX code will not be sufficient to ensure widespread deployment (even if users wanted to cooperate). If the correctness of the network design depends on having a single, universally deployed standard, it will be extremely difficult to update that standard; smooth network evolution requires the assumption of multiple versions deployed at the same time.

We have moved into an era of commercial networking. While the Internet started out as a research network, it has gone through a rapid transition to a commercial service.

Second, the diversity of applications on the Internet is ever-increasing. Best-effort service can be used for packet voice and video (such as *vat* and *nv*) and reliable multicast [14] (such as is used in *wb*) in addition to its more traditional uses of file transfer, electronic mail, and remote login. It no longer makes sense to artificially confine users to a single acceptable congestion control algorithm when their needs differ.

Third, and most important, we have moved into an era of commercial networking. While the Internet started out as a research network, it has gone through a rapid transition to a commercial service. Other network services, such as asynchronous transfer mode (ATM), have been designed from the start as international public telecommunication services, and have thus been "commercial" from conception. The very nature of best-effort service precludes specifying the actual packet delays a user will experience. However, users of a commercial network are unlikely to accept having the service they receive depend on the "polite" behavior of other users.

Thus, the basic paradigm of congestion control for best-effort service must be reformulated to suit the new context of commercial public networks. Our congestion control paradigm must be built on the following two basic principles. The first principle, which was originally articulated in connection with work on fair queuing [15, 16], states that the adequacy of service delivered to a particular user should not depend on the detailed behavior of other users. Clearly, the bandwidth available to a particular user must depend on whether other users are also trying to use the network at the same time, but the network must try to isolate the service provided to a user from the details of how other users respond to congestion. This principle renders the issue of cooperation moot; the service one user gets should not depend on whether or not all users cooperate. Providing isolation entails enforcing limits on the resources made available to users. Resources should not be given out on a first-come, first-served basis; the network must more actively manage those resources.

The second principle is that the network should provide enough feedback to users that they can effectively utilize the available resources. If the network is enforcing resource limitations, it should provide users with enough information about those limitations so that each user can use her own share of the resources effectively. This requires that users have a model of how the network is behaving; that is, we need a service model, or specification, for best-effort service.

The reformulation of congestion control for best-effort service according to these principles is the subject of this article. We are not attempting to design specific new congestion control algorithms. Instead, we are merely trying to articulate the design principles. Many of these principles have been discussed in other forums; however, with ATM currently designing a best-effort service under the name Available Bit Rate (ABR) and the increasing commercialization of the Internet, these issues warrant revisiting.

In the next section we outline the service model for best-effort service. In the third section we describe the set of mechanisms available to implement this service model and contrast their various roles. The fourth section articulates the implications of our findings for future network design. We conclude, in the fifth section, with a discussion of some well-known examples of congestion control mechanisms.

Service Model

The service model is an abstract definition of the service that a network client will receive. It defines a long-term contract between the network and the application writer by defining a stable interface; details may change, but the semantics of the service cannot. Thus, the service model documents the commitments a network makes to a set of clients when they request that service.

In the past the Internet community has not specified the service model for best-effort traffic. Internet service has only been defined operationally, and network applications are designed to be flexible enough to not need a particularly well-defined service.

Currently, the Internet's best-effort service is provided by first-in first-out (FIFO) queuing and tail-first dropping (last-in first-dropped) in routers, but there is nothing in the specifications that prevents other queuing strategies from being used.

On the other hand, because of its commercial nature and demanding service requirements of the deployed end-user equipment, the telecommunications industry has historically written service requirements and models early in the process of defining a new service. This difference is largely due to the fact that the telecommunications industry started with a specific application (i.e., telephony) and built a network to suit it. The Internet, on the other hand, started in exactly the opposite way: it started with a new network technology and explored, successfully, new applications that were able to use the undefined service.

As the telecommunications industry begins to use ATM as its infrastructure, it is also moving to incorporate best-effort-like services in ABR. Therefore, it is in the context of ATM that the first attempt to write down a service specification for a best-effort traffic class is being made. As of Autumn 1995 the first phase of this work was mostly done [17], but considerable detail remains to be finalized. The resulting service model has sufficient generality to be useful to the Internet community. The service model described below served as the basic input to the development of the ATM service model described in I.371 [17] but is less formal than the I.371 text.

We now describe the broad outline of a service model for best-effort traffic. The first issue to consider in a service model is the requirements of the applications which will use best-effort services.

These requirements form the basis of the service requirements. We claim that the delay requirements are *as soon as possible* (ASAP), and corresponding bandwidth requirements are *as much as possible* (AMAP). Applications that utilize best-effort service can typically use data as soon as it arrives. Although many of them, such as remote login, desire a short delivery delay, nevertheless they always use the data, even when the data has suffered from a long network delay; that is, these applications do not have quantitative real-time constraints. Similarly, applications that use best-effort service can typically use as much bandwidth as is available, yet are capable of making progress with as little as they are given. If the application completes sooner due to being given additional bandwidth, the user's satisfaction will increase; the sooner the application completes, the better. This is in sharp contrast to applications such as voice transmission, where failure to receive specified service amounts and delays may result in application failure, and the application is intrinsically incapable of using additional bandwidth even when it is available. In short, the bandwidth and delay requirements of best-effort applications are *elastic* or scalable.

Because the bandwidth requirements are scalable, users may dynamically share available bandwidth. This leads to a major difference between best-effort service and previously existing telecommunications circuit services: the bandwidth available to the user, and the delivery delay of their packets, varies moment to moment. More important, there should not be an admission control to this service [18]. Consequently, there should not be any quantitative bounds to the service; there are no lower bounds on bandwidth and no upper bounds on delay.

What does it mean to offer a service model when the network makes no quantitative assurances? There are two kinds of commitments the network can make: relative and procedural. Relative assurances are those in which the network ensures that the bandwidth received by those flows that share the same path or the same bottleneck point is fairly apportioned,[4] according to some commonly agreed-upon definition of fairness.[5] Why include such a relative assurance of fairness? There are several reasons for this. The first is that, with a best-effort service, the user is not purchasing a quantified service, but rather a procedure for delivering service. Customers and regulatory bodies are likely to insist that this procedure be fair. The second reason is that there seems to be a strong link between network stability and fairness. While it is not known if fairness is a necessary condition for stability, it is known to be a sufficient condition [26].

It is in the context of ATM that the first attempt to write down a service specification for a best-effort traffic class is being made.

Procedural assurances tell users what they can expect if they respond to the network's congestion signals appropriately. Although the absence of admission control makes it impossible to ensure quantitative delay or throughput in service, there are still possibilities to control packet losses. A user can achieve a low packet loss rate if the network provides feedback to inform users of the bandwidth actually available at that instant in time. A procedural assurance takes the form "if you use the service in this particular manner then you will not lose (many) packets." While the network cannot promise how many packets it will deliver or when it will do so, it can make the promise that if the source behaves in a certain way in response to network conditions, then no (or few) packets will be dropped due to congestion. Thus, the form and content of that feedback compose the most important element of the service model. The relative assurances described above mean that two users sharing the same path and same weights will receive similar feedback content.

Taken together, these relative and procedural service model elements define a service which operates via a *closed-loop control*. The users can only achieve good performance when they participate in the closed-loop control system. Of course, a full service model will involve much more detailed specifications. Interested readers are referred to [20] for the evolving ABR service definition in the ATM community.

Mechanisms

To support the service model, the network must actively manage its own resources and also provide feedback to users. Users must respond effectively to these congestion signals. What mechanisms can we use to accomplish these goals?

In this section we discuss four orthogonal aspects of congestion control mechanisms out of which a best-effort service can be built: packet scheduling, buffer management, feedback, and end adjustment. We believe it highly likely that elements from all four of these aspects are required for a complete description of a best-effort mechanism. Our purpose here is to delineate the different roles these aspects play in providing effective congestion control for best-effort service.

Scheduling

There are two ways in which the network can actively manage its own resources: packet scheduling and buffer management. When the instantaneous load is very light and there is little, if any, queuing of packets, the scheduling and buffer management algorithms show little effect. When there are sizable queues, however, these two mechanisms control which flows get access to bandwidth and the buffer. Thus, in what follows we will focus mainly on the behavior of these mechanisms under high instantaneous loads.

Scheduling controls the order in which individual packets get served (or whether they get served at all). Scheduling is the most direct control over how the network serves every user. We furthermore believe it is the only effective control. As we argue below, buffer management alone cannot provide flexible and robust control of bandwidth usage. Moreover, if we use scheduling effectively, the buffer management algo-

[4] *Fairness does not necessarily imply equal portions. Weighted fairness provides network operators the ability to apportion bandwidth on the basis of policy considerations. I.371 uses the term "defined allocation policy" instead of "fairness" to emphasize this point. Fairness, however, does imply that there will not be unequal allocation for non-policy reasons.*

[5] *Relative assurances can also apply to the assurance that higher priority levels will receive better service (even though no level is given a quantitative assurance).*

rithm need not be precisely tuned.

Two of the most popular scheduling algorithms are FIFO and weighted fair queuing (WFQ) [16]. FIFO serves packets in the order of arrival. Due to its simplicity, FIFO scheduling can be found in most of today's network implementations. With FIFO, all users experience the same delay on average, even if the overload is caused by a small subset of the users; thus, it provides no protection against uncooperative users. Worse yet, it may also lead to pathological unfairness even when all end users behave properly; this phenomenon of flow segregation is described in [5, 16]. FIFO scheduling may also create packet clumps; such clumps are formed when a series of packets in a particular flow enter the switch well separated but then, because they are queued up and there have been no other intervening arrivals, the packets leave the router in a back-to-back clump. This form of clumping can cause congestion at downstream routers.

Fair queuing, or any of its rough functional equivalents such as round-robin, attempts to split the bandwidth evenly among the currently present flows; see [16, 21–23] for more extensive discussion of such scheduling algorithms. This active management of bandwidth provides each flow with a great degree of protection from other flows. FQ also avoids or reduces the packet clumping problems; because of its round-robin-like behavior, FQ interleaves packets from competing flows. This can be thought of as putting a flow through a low-pass filter.

Because network load can change rapidly, packets are likely to be dropped whenever a surge of packets occurs unless adequate buffer space is available to temporarily harbor the packets.

We are not saying that all switches must implement FQ, only that they must implement some scheduling algorithm that can enforce a bandwidth allocation policy. Otherwise, greedy users can capture more than their share of the bandwidth.

However, scheduling alone does not prevent substantial packet losses. Because network load can change rapidly, packets are likely to be dropped whenever a surge of packets occurs unless adequate buffer space is available to temporarily harbor the packets. Thus, buffer management algorithms are an important part of congestion control; we discuss them in the next section.

Buffer Management

Buffering is required at a switch whenever packets arrive faster than they can be sent out. However, if the packet overload persists for a long enough period, the switch's (necessarily finite) buffering capacity will eventually be exceeded, and packets must be discarded. The role of buffer management is to select which packets get dropped in such an overload situation. Our best-effort service model requires that a user who responds appropriately to the congestion feedback given by the network will see a low loss rate. When the bandwidth available to a user decreases, the system must provide sufficient buffering so the user has a chance to reduce his or her load. Thus, buffer management is an important aspect of congestion control. Two of the most popular schemes for buffer management are shared buffer pool and per-flow allocation.

The shared pool approach aggregates the buffer requirements of all flows into one buffer pool. It serves packets in a first-come first-use (FCFU) way, and packet dropping is based on the occupancy level of the shared pool. Due to its simplicity, the FCFU pool can be found in most implementations today. Shared pool buffer management clearly does not protect flows from each other; a single flow can occupy all of the buffers, causing other flows to be denied service.

Schemes that allocate buffers on a per-flow basis keep track of the buffer utilization of each flow and drop packets based on the occupancy level of an individual flow's allocated buffers. Per-flow buffer management schemes protect well-behaved flows in terms of buffer usage. Knowing the amount of buffering available can help the flow determine how it should respond to congestion signals in order to prevent excessive packet drops. There are a wide variety of possible policies on buffer allocations; in addition to depending on the number of other flows present, the buffer allocation might depend on the current allocated bandwidth and the previous buffer allocation. Such sophisticated buffer allocation policies are probably the least well understood aspect of congestion control mechanisms.

Feedback

Feedback from the network to the network client provides the client with information necessary to adjust to changes in the available bandwidth. Over the years there have been many different mechanisms described for feedback. What considerations are relevant when designing the feedback mechanism?

One important factor is multicast data delivery — the delivery of data from one or more senders to a group of receivers. Multicast does not imply significant design changes for scheduling or buffer management, but it is relevant to feedback mechanisms. In particular, two issues that arise in the context of multicast are scalability and policy. How does one avoid feedback implosion at the source? Should the feedback mechanism have embedded within it the policy on how to respond to different congestion levels along different branches in the multicast path?

Also, in designing a network that will be heterogeneous with respect to switch design and manufacture, it is architecturally important to separate the manner in which feedback is given from the precise mechanisms used in the switches. That is, we should separate the interfaces used to communicate feedback from the implementation of that feedback signal.

With these design issues in mind, in this section we will try to provide a taxonomy for these congestion feedback mechanisms and discuss some of the issues that should be addressed when a particular mechanism is chosen.

Feedback can be either explicit or implicit. Explicit feedback uses an explicit indicator or field in the packet stream to convey network status. An example of explicit feedback is the DECBit protocol [1], in which a single bit of information is conveyed by the network from the sender to the receiver, where it is the receiver's responsibility to return the information to the sender.

In contrast, implicit feedback requires end users to monitor the performance of their data transmission for clues to the current network status. The most well-known example of implicit feedback is the TCP Slow-Start algorithm, which uses packet drops as implicit feedback of congestion. Two other examples of implicit feedback are the packet pair protocol [24], which uses the absolute delay dispersions of packets exiting the network as feedback on the maximum rate at which the network could provide service, and NETBLT, which compares the observed throughput with the transmission rate to determine the maximum achievable throughput [25].

We now discuss the implicit and explicit approaches in more detail.

Implicit Feedback — Packet dropping is the most common form of implicit signal. The dropping of a packet does not necessarily have to be an indication of buffer exhaustion. Algorithms such as random early drop (RED) use packet drops to signal the approach to congestion. Note that unless the end host knows which dropping algorithm is being used in the switch, it cannot distinguish between RED's indication of the onset of congestion and a FIFO switch's indication of buffer exhaustion, and thus may not be able to appropriately choose a control strategy.[6]

Multicast does not imply significant design changes for scheduling or buffer management, but it is relevant to feedback mechanisms.

Another implicit feedback that has been used is observation of the rate at which packets emerge from the network. This works if the bottleneck rate is reflected in the packet stream rate at the egress point. For small numbers of packets the signal may be very noisy, thus requiring averaging over intervals. A key issue is the length of the time interval over which averaging occurs. If this interval is larger than the interval over which feedback must be returned to the sender, problems due to control delay will occur. The scheduling algorithms at the switches interact strongly with the necessary averaging intervals.

A final example of implicit feedback is measurement of the end-to-end delay change as one changes the transmission rate (e.g., TCP Vegas [2] or delay-based congestion control [1]). With FQ switches, delay will be relatively constant until the fair-share bandwidth allocation is reached, at which time queuing will occur, and a sharp increase in delay with no increase in bandwidth will be observed by the application. However, such an approach may not work very well in a network with FIFO switches, where one can still observe an increased throughput by increasing the transmission rate, even when the queue builds up and total delay increases.

The chief advantage of implicit feedback over explicit feedback is that with implicit feedback the network only needs to focus on resource allocation, but does not have to calculate a precise control signal. End users can always derive implicit feedback from the observed throughput, delay, and packet losses. The question is whether one can derive correct and accurate information from the perceived performance. For example, as in the case of flow segregation, an implicit feedback signal can be misleading; thus, a prerequisite for implicit feedback is a well-defined policy for resource allocation — the scheduling control and buffer management we discussed earlier — used by all switches in the network.

Explicit Feedback — Explicit feedback can be given in either the forward or reverse directions. In forward feedback the packet stream is marked as it traverses the network from sender to receiver. It is the receiver's responsibility to return the feedback signal to the sender. In reverse (or backwards) feedback the signal is carried directly from the switches back to the sender in a reverse flow. In the Internet protocol (IP) suite a Source Quench ICMP message is a form of backwards feedback. Frame relay has both a forward explicit congestion notification (FECN) indication and a backwards explicit congestion notification (BECN) message.

Note that BECN does not fit the multicast model very well. If the congestion signals from each node are sent independently to the source, we have an implosion of such control messages, which does not scale well for large multicast groups. If these congestion signals are merged by the network along the reverse path, the network has to embody the policy on how such congestion signals should be combined. Neither approach is satisfactory. The FECN approach leaves it to the application to determine how best to merge such congestion information.

Explicit feedback signals can also be either binary (i.e., "congestion experienced") or multivalued (i.e., "how much congestion has been experienced"). If the feedback is binary, or restricted to only a few values, the end-systems must guess the quantitative significance of the feedback. With binary feedback the network specifies when it is congested but not how badly. This leads to asymmetrical behavior on the part of senders: rapid drop of the transmission rate to an "estimated" safe value when congestion is indicated, followed by slow increases when the congestion feedback is no longer set.

Explicit feedback requires the switch to calculate the feedback associated with a flow and to inject that feedback into the packet stream. Thus, it differs from implicit feedback, where it is the effects of the switching elements on the packet stream that are being measured. Explicit feedback requires an extra mechanism. However, one advantage with explicit feedback is that it can provide more quantitative control information to end users and thus speed up the adjustment process to load changes.

End Adjustment

The end adjustment algorithm is heavily dependent on the form of feedback and the quantities being controlled within the network. If the feedback information from the network is imprecise or incomplete, it may even be necessary to search for the correct operating point. For example, with a binary feedback signal the correct operating point is found through an iteration process of network feedback and end-system adjustments.

The end-system adjustment is the response to a servo control loop. For rate-controlled systems this servo loop needs to match the rate at which the source is sending to the rate at which the network bottleneck can carry packets. In addition, when a temporary overload occurs it must allow queued data to drain out of the bottleneck node.

The precision of the servo loop determines network performance. If switch buffers are allowed to become empty, one may not reach the maximum possible throughput. Keeping some data in switch buffers keeps the throughput high since the line would never go idle even in the presence of feedback

[6] *It is not clear that there is a difference in the appropriate response in these two cases (RED and FIFO), but there are presumably other possible dropping strategies where the difference is more important.*

delay. On the other hand, if the buffers become too full, packets will be discarded by the buffer management algorithms.

As discussed previously, the network must not depend on homogeneous implementation of a single adjustment algorithm at all user ends. We should note that heterogeneity will also arise because individual users are likely to react differently to the same feedback signal because different applications have different sensitivities to delays and losses. For instance, if an application is designed to tolerate high losses (e.g., the image transfer protocol designed by Turner and Peterson [26]), it is unlikely to react to feedback in order to reduce the loss rate as quickly or aggressively as a loss-sensitive application would.

Beyond issues of loss, best-effort multicast service also raises a new issue: how to handle a multivalued feedback. Consider a multicast network where one receiver has a much slower network link than all the others. Is the correct source response to this to slow down, or to continue at the rate that the other receivers can handle? Or, is it to layer-code the data and expect bottleneck switches to selectively discard based on the layer coding? The answer is application-specific and thus should not be built into the network's calculation of feedback.

In summary, there is not a single expected or acceptable end behavior. The network should not have a fixed expectation for end-system behavior. On the other hand, the network must have a defined behavior on which the end user can count. Otherwise, the network response to end-system adjustments will not be predictable, and the application cannot use adjustments to implement policy or even to control loss.

We believe that packet scheduling, buffer management, feedback, and end adjustments are all necessary components of any best-effort network design.

Why We Believe that All Four Are Necessary and Sufficient

We believe that packet scheduling, buffer management, feedback, and end adjustments are all necessary components of any best-effort network design, and we speculate that they are sufficient components for providing best-effort services. Scheduling and buffer management are the methods by which the network and each individual switch enforce policy. Feedback is the method by which switches inform the end-system about the state of the network. End-system adjustments close the control loop and implement application-specific policies.

Scheduling controls the rate at which users gain service and ensures that a user transmitting at its share of the bandwidth will not be denied service by a user who is not. A more sophisticated scheduling scheme than FIFO must be present to enforce sharing. Share enforcement cannot be done by buffer management alone because with FIFO, a user's share of the bandwidth is proportional to the number of packets queued in the FIFO queue. Buffer management without some form of scheduling beyond FIFO can give fair shares only when each user is allocated the same share of the buffer, and the users all keep their share of the buffer *completely full* all the time. Unfortunately, keeping the buffers full results in high packet loss rates. Thus, buffer management cannot be a replacement for scheduling control.

Buffer management is necessary because even the best packet-scheduling algorithms do not by themselves protect against abusive users. For example, WFQ scheduling may guarantee a fair share of the bandwidth of the outgoing link, but it does not guarantee buffer space. Consider a switch with two users, A and B, each on a separate input link. Assume that A and B's fair share of the bandwidth is 50 percent (unweighted). Even if A is transmitting exactly at its fair share, B could be transmitting at 100 percent. Absent any buffer management, B will fill up all the buffers in the switch. Now assume that A's share of the bandwidth decreases because a source C comes on-line. By using up all the buffers, B will have denied A the ability to track changes in the available bandwidth without losing packets.

The precise allocation of buffers is dependent on the scheduling algorithm used. As scheduling algorithms depart from a fluid flow model (e.g., WFQ), clumping can occur in the packet flow. This clumping must be allowed for by the buffer allocation algorithm.

Feedback informs the end users of network conditions and allows the users to calculate their current fair shares. If the network provides unfair or inconsistent feedback to the user, a variety of pathologies will occur. Thus, feedback is a necessary component in a network design. However, feedback cannot force the user to actually obey it. Thus, feedback must be supplemented by enforcement mechanisms such as scheduling and buffer management.

End adjustments close the control loop. They are the response to network feedback. Poor-quality adjustments will potentially result in high loss rates or low network utilizations. However, just as the network cannot count on end users to obey feedback, the network cannot count on users to implement any specific form of end-system adjustment. Network integrity must depend on switch-resident mechanisms (i.e., packet scheduling and buffer management), not end adjustments; and end adjustments cannot work well if the quality of the feedback is poor.

Implications for the Design of Networks

In this section we will argue that carefully thinking through the roles of the four mechanisms is essential if we are to build a network which has long-term evolutionary potential. In particular, we must ask the architectural question of "what belongs in the center of the network (i.e., in the switches) and what belongs at the edge of the network (i.e., in the terminal equipment)?"

Equipment at the center of a network is usually exceptionally stable. This is due to the operational constraints of running a production network; the network operator cannot run the risk of either destabilizing an operational network or desupporting running applications. As a result, any new application that requires an upgrade to the core of the network is likely to have very long introduction times. Thus, we should consider the core of the network as requiring long-term stability and plan for evolution to occur at the periphery.

Of the four mechanisms, two are likely to be quite stable and one mostly stable. Scheduling may take many forms and implementations (e.g., WFQ, round-robin, token buckets, and/or approximations to these mechanisms), but there are deep theoretical similarities between all rate-allocating schedulers, leading one to believe that the exact form of scheduling may be less important than the presence of scheduling. Changing the form or content of the feedback implies a fundamental redesign of the network and is therefore unlikely.

Thus, we expect scheduling and feedback to be quite stable.

Since buffer allocation must respect control loop delays, buffer management requires both knowledge of the average time delays in the control loop and the average rates at which a flow is legally transmitting. It can also make good use of knowledge of traffic statistics. Over time it is likely that implementations will change estimating techniques for delay bandwidth products and will utilize improved estimates of traffic fluctuations, thus changing the absolute amount of buffering allocated to a flow. However, changes in the amount of buffering allocated to a flow primarily affect the loss rate on that flow during periods of congestion. Thus, one would expect that buffer management policy can experience some change without adversely affecting end users.

It is when we consider end adjustments (i.e., end-system control of the servo loop) that we find an area where rapid and significant change is both possible and desirable. When a network is robust there is no need to enforce a specific behavior on the user. Users can change both the parameters of the servo loop and the algorithms. Parameter changes include, for example, changes to the rate at which the user increases her or his sending rate and the amount of data in transit at any given time. An example of an algorithm change would be new methods of more precisely estimating the evolution of network state based on prior feedback. Robustness enables and encourages experimentation and evolution in end adjustments.

It is when we consider end adjustments (i.e., end-system control of the servo-loop) that we find an area where rapid and significant change is both possible and desirable.

If the most likely sources of technical change are the adjustments of the servo loop, adjustments should be made only at the periphery (e.g., at end-systems). Switches should only provide feedback, not participate directly in the control loop calculations. Although the previous statement sounds simple, it tends to rule out hop-by-hop schemes, especially those where the network can run either end-to-end or hop-by-hop, and feedback schemes that rely on the switches modifying feedback based on control-loop-like considerations.[7]

Examples

In this section we briefly describe several well-known congestion control algorithms, and explain where they fit into our taxonomy discussed earlier in the article.

Current Internet

Until recently, the traffic across the Internet has been dominated by data applications, with electronic mail, remote login, and file transfer being the main sources of data. These data applications run on top of the reliable TCP transport protocol [27]. Only recently, due to audio and video transmissions on the multicast backbone (MBone) [28], has user datagram protocol (UDP) traffic become a significant factor in Internet traffic.

One strength of today's Internet is TCP's extremely well-designed adaptive retransmission and congestion control mechanism (Slow-Start), which was designed and implemented by Van Jacobson in the mid-'80s [6]. When Slow-Start was designed the Internet had a deployed base of routers with FIFO scheduling, FCFU buffer management, and (as a result) drop-tail service. Thus, TCP Slow-Start uses the packet losses from the network as an implicit signal for network congestion. Whenever packet losses are observed, a sender immediately reduces its data transmission rate to one packet per round-trip time, and gradually opens up the congestion control window only if no further losses are observed. The increase in transmission rate is divided into two phases: congestion recovery, during which the window size opens up rapidly, and congestion avoidance, during which the window size increases by only one packet per round-trip time. Universal deployment of this conservative approach together with improved network capacity have kept the Internet from congestion collapse even in the face of exponential growth in both the user population and the traffic load.

We note that the feasibility and effectiveness of Slow-Start congestion control relies on the homogeneity of the transport protocol and, probably more important, the cooperation of essentially *all* end hosts. When one or a few end hosts do not follow the rule, they can grab as much network bandwidth as they like while other, obedient users back off from congestion. The growing UDP traffic from MBone applications, which do not implement Slow-Start, provides a challenge to the previously homogeneous end-host behavior; recently, a token-bucket mechanism has been added to all MBone routers to limit the total volume of MBone traffic.

According to our taxonomy, the approach of the current Internet can best be described as no scheduling control of allocations, no buffer management control of allocations, implicit feedback, and cooperating end hosts. As pointed out in [6], however, end-host adaptation can only be part of the story. To minimize congestion losses and provide fair services, network switches must also be engaged in the control. The recent RED work [3], which we describe next, is one step forward in that direction.

RED

The RED congestion control algorithm design retains the basic design of today's FIFO routers. It uses the average queue length as an indicator of network load, but no other state information is kept concerning the resource utilization of individual flows. Incoming packets are randomly dropped as soon as the potential of overloading exists, well before the switch reaches buffer pool exhaustion. RED provides implicit feedback via dropping and thus does not require any changes to the existing end implementation of TCP Slow-Start.

Because packets to be dropped are randomly chosen from the incoming data stream, flows sending at faster rates will incur more packet losses than flows sending at slower rates (but all flows experience the same packet loss rates, in that the number of losses for each flow will be, on average, proportional to the number of packets trasmitted by the flow).

[7] *An example of such a switch involvement in control-loop calculations is a scheme such as has been proposed by the ATM Forum in which the sole feedback is rate. As a result, the switches have to modify the rate feedback based on buffer occupancy — a calculation which embeds control-loop functionality in the rate feedback.*

However, flows which do not implement Slow-Start can capture more than their share of bandwidth (i.e., have more packets forwarded). Therefore, this stateless scheme, by itself, does not provide effective resource usage enforcement. Reference [3] suggests that, in the presence of persistent heavy load despite RED, the switch can identify specific sources as offenders from the number of dropped packets and treat them specially (e.g., dropping all of their packets) — that is, to enhance the switch with some per-flow state.

By our taxonomy, one may consider RED as both an indirect method of scheduling control and a buffer management strategy, with implicit feedback and the Slow-Start end adjustment algorithm. RED does not ensure fair sharing of resources. With the enhancement of per-flow state for misbehaving hosts, RED may function adequately as long as the vast majority of users adopt the same congestion response algorithm.

Hop-by-hop credit schemes must do explicit buffer management since the feedback is in terms of the space left in the buffer.

DECbit

The DECbit congestion control scheme predates Slow-Start [24]. It is an end-to-end explicit feedback control scheme with a binary value signal, the congestion indication bit. The goal of DECbit design is congestion prevention. The performance of the network is measured by "power," which is defined as the ratio of throughput over delay, which achieves its maximum value when the link stays busy all the time and the queue size is kept to one. Therefore, in DECbit the switches measure the average queue length, and set the congestion indication bit upon a packet arrival whenever the average queue length exceeds one. Switches along the path may set the bit, but can never clear the bit once it is set.

The receiving end sends the bit in each data packet back to the sender in the acknowledgment packet for that data packet. The sender then examines the bit in the last W packets, and reduces the flow control window size if at least 50 percent of the packets have the bit set; otherwise, the sender increases the window. The window size adaptation uses a multiplicative-decrease with additive-increase scheme, aiming at fast adaptation to congestion and incremental speedup in the absence of congestion. However, because the multiplicative factor used in the decrease is large (0.875), the actual adaptation speed is rather slow as compared to the radical reduction taken by Slow-Start upon a packet loss.

As one can see, DECbit is very similar to Slow-Start. In our categorization, DECbit has no scheduling control and a shared buffer pool management strategy (i.e., keeping the average queue from exceeding one). The most noticeable difference is its explicit congestion bit, as opposed to implicit feedback in Slow-Start or RED. Through simulation the DECbit designers noticed unfair service due to the stateless control even when all end hosts obey the control rules. A bit-setting scheme based on per-flow measurement has been proposed as a fix to this unfairness [29]; but even with that fix, the entire control scheme relies on cooperation from all end hosts.

The Hop-by-Hop Credit Scheme

Hop-by-hop credit [7, 8, 30] differs from the three examples above in three important ways. First, hop-by-hop credit bases its controls and feedback on the amount of space left in switch buffers. Feedback is explicit and multivalued: the number of unused units worth of buffering. Second, hop-by-hop credit always attempts to utilize all available unused units of buffering. Hop-by-hop attempts to keep the buffers completely full instead of mostly empty. Third, it segments the control loop at each switch instead of running an end-to-end control algorithm. This segmentation is forced because the control algorithm attempts to keep the buffers full; if an end-to-end control algorithm tried to keep buffers full, buffer overflow would inevitably happen. However, hop-by-hop credit schemes tend to get very good network utilization, even in the face of widely varying traffic loads, because there is usually always data in the switch buffers which can be sent when an opportunity arises.

Hop-by-hop credit schemes require per-flow queuing to avoid both head-of-line blocking and deadlock. Once per-flow queuing is in place, a method of scheduling is required; the algorithms in [7, 8, 30] use round-robin scheduling. A side effect of this scheduling is that flows are given fair service.

Hop-by-hop credit schemes must do explicit buffer management since the feedback is in terms of the space left in the buffer. Reference [30] describes a scheme in which each flow is given a static allocation of buffer memory. Unfortunately, such a static allocation, while simple, leads to large memory requirements in the wide area. Subsequent schemes have been developed [15, 16] that do dynamic allocation of buffer space. Dynamic allocation seems to require knowledge of the rate at which a connection is being serviced. It also introduces statistical sharing of buffers, and therefore the possibility of loss.

There are two potential issues with hop-by-hop credit schemes. The first is that in order to achieve good throughput, buffers are kept full, with the concurrent possibility of increased transmission delay within the network. The other potential issue is that the switch-by-switch control loops embed control policy within the hardware of the network. In case of multicast data delivery, for example, it enforces a policy of everyone adjusting to the slowest receiver, leaving no option for the application to decide on a different policy.

A Strawman Proposal

In a contribution to American National Standards Institute (ANSI) Committee T1S1, one of the authors of this article describes a mechanism that was built using the insights described in this article [31]. This mechanism uses WFQ as the scheduling mechanism, buffer allocation based on the user's allocated bandwidth, forward explicit feedback of the bottleneck rate and buffer utilization, and an end-system adjustment that tracks the bottleneck rate but allows for overly full buffers to drain.

Effectively, the mechanism takes the packet pair protocol of Keshav [24], which uses implicit feedback (dispersion of packets, packets in transit) to allow the destination to calculate the bottleneck rate and the buffer occupancy in the network, and makes what had been implicit explicit. This explicit data is carried by special ATM cells called "resource management (RM)" cells. The values contained in the RM cells are modified as the cells traverse the network.

The combination of bottleneck rate and bottleneck buffer occupancy allows the end adjustment to carefully control the bottleneck queue length, thus ensuring that packets are not lost due to queue overflows and that the user does not lose throughput by letting the bottleneck queue go empty. Because [24] did not address buffer management (although it was clearly understood to be an issue), it was not possible to show robustness and stability in the presence of ill-behaved users.

While the scheme in [31] is not known to be optimal in any sense of the word, it does provide near 100 percent link utilization without packet loss for the configurations and loads for which it has been simulated. The intent of the contribution to T1S1 was to demonstrate to the ATM community the potential performance of a carefully designed end-to-end rate-based feedback scheme which also had substantial theoretical grounding. For the purposes of this article it demonstrates how explicitly thinking about the service model and mechanism "knobs" of our best-effort service gives us insight into mechanism design.

The current congestion control paradigm assumes that end users will use a single mandated algorithm. While the work done in this area has proven to be of great value, we need to recognize that this paradigm is clearly inappropriate for future public networks.

Summary

The current congestion control paradigm assumes that end users will use a single mandated algorithm. While the work done in this area has proven to be of great value, and is extremely interesting, we need to recognize as a community that this paradigm is clearly inappropriate for future public networks. There are many reasons for this. First, we cannot count on user cooperation in such public networks. Second, given that there are many uses of best-effort traffic with very different delay and drop sensitivities, it makes little sense to mandate a single algorithm. Third, in a commercial network users will not accept a service so dependent on the detailed behavior of other users.

Thus, we must turn towards a paradigm that is not built on the assumption of homogeneity. One principle underlying this new paradigm is that the network should enable users to achieve good service. This requires that the network provide sufficient feedback so end users can use the available bandwidth effectively. This, in turn, requires that there be a well-defined service model which describes the nature of the service provided. Another fundamental principle is that the network, in delivering service to a particular user, must not count on cooperation from other users. This requires that the network protect flows from each other by enforcing restrictions on resource usage. We believe packet scheduling is the most effective tool for this enforcement.

It is through the consideration of both network evolution and the types of mechanism available to support best-effort traffic that we may derive a very general design principle: the network should provide isolation of flows and give the best explicit feedback possible, while the end-systems should implement the control function.

These points are very simple, but they reflect a fundamental change from the current discussions of congestion control. Our purpose in this article is not to design the details of future algorithms, but to change the nature of the debate about those algorithms.

Acknowledgments

Many people have provided feedback on earlier versions of these ideas. We would like to thank our colleagues in the ANSI/International Telecommunications Union (ITU) ATM community for their feedback on the service model. We would like to explicitly thank Kevin Kahn and Tom Anderson for their feedback.

This research was supported in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contracts DABT63-94-C-0073 (SS,LZ), DABT63-92-C-0034 (BL, CL), and DABT63-92-C-0002 (CL). The views expressed here do not reflect the position or policy of the U.S. government.

References

[1] R. Jain, "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *Comp. Commun. Rev.*, vol. 19, no. 5, 1989, pp. 56–71.

[2] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New Techniques for congestion detection and avoidance," *Proc. SIGCOMM '94*, Aug. 1994.

[3] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. on Networking*, vol. 1, no. 4, Aug. 1993.

[4] A. Charney, "An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback," Master thesis, MIT Laboratory for Computer Science, May 1994.

[5] S. Floyd and V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways," *Internetworking: Res. and Exp.*, vol. 3, no. 3, Sept. 1992.

[6] V. Jacobson, "Congestion Avoidance and Control," *Proc. SIGCOMM '88*, Aug. 1988.

[7] H. T. Kung, T. Blackwell, and A. Chapman, "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistical Multiplexing," *Proc. SIGCOMM '94*, Aug. 1994, PP 101–15.

[8] H. T. Kung and K. Chang, "Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks," *Proc. Infocom '95*, April 1995, pp. 239–52.

[9] K. K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM Trans. on Comp. Sys.*, vol. 8, 1990, pp. 158–81.

[10] S. Shenker, L. Zhang, and D. Clark, "Some Observations on the Dynamics of a Congestion Control Algorithm," *ACM Comp. Commun. Rev.*, vol. 20, no. 4, Oct. 1990, pp. 30–39.

[11] R. Wilder, K. K. Ramakrishnan, and A. Mankin, "Dynamics of a Congestion Control and Avoidance of Two-Way Traffic in an OSI Testbed," *ACM Comp. Commun. Rev.*, vol. 21, no. 2, Apr. 1991.

[12] L. Zhang and D. Clark, "Oscillating Behavior of Network Traffic: A Case Study Simulation," *J. Internetworking: Res. and Exp.*, vol. 1, 1990, pp. 101–12.

[13] L. Zhang, S. Shenker, and D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," *Proc. SIGCOMM '91*, 1991.

[14] S. Floyd *et al.*, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *Proc. ACM SIGCOMM '95, Comp. Commun. Rev.*, vol. 25, no. 4, 1995, pp. 342–56.

[15] J. Nagle, "On Packet Switches with Infinite Storage," *IEEE Trans. on Commun.* vol. 35, 1987, pp. 435–38.

[16] A. Demers, S. Keshav, and S. Shenker. "Analysis and Simulation of a Fair Queueing Algorithm," *J. Internetworking: Res. and Exp.*, vol. 1, 1990, pp. 3–26; also in *Proc. ACM SIGCOMM '89*, pp. 3–12.

[17] ITU-T, Study Group 13, "Draft Recommendation I.371," Geneva, Switzerland, July 1995.

[18] S. Shenker, "Fundamental Design Issues for the Future Internet," *IEEE/ACM Trans. on Networking*, vol. 13, no. 7, Sept. 1995.

[19] S. Shenker, "A Theoretical Analysis of Feedback Flow Control," *Proc. SIGCOMM '90*, 1990.

[20] Bryan Lyles, ed., *ABR Baseline Document T1S1.5/94-005R3*, revision R3 dated Oct. 14, 1994. Releases of the document prior to Oct. 14, 1994 were entitled *Class-Y Baseline Document*.

[21] M. Katevenis, "Fast Switching and Fair Control of Congested Flow in Broadband Networks," *IEEE JSAC*, vol. 5, 1987, pp. 1315–26.

[22] S. Morgan, "Queueing Disciplines and Passive Congestion Control in Byte-Stream Networks," *Proc. Infocom '89*, 1989, pp. 711–20.
[23] E. Hahne, "Round-Robin Scheduling for Max-Min Fairness in Data Networks," *IEEE JSAC*, vol. 9, 1991, pp. 1024–39.
[24] S. Keshav, "A Control-Theoretic Approach to Flow Control," *Proc. SIGCOMM '91*, Sept. 1991, pp. 3–15.
[25] M. Lambert, "An End-Point Adaptive Rate Control Strategy for the NETBLT Protocol," preprint, 1988.
[26] C. Turner and L. Peterson, "Image Transfer: An End-to-End Design," *Proc. SIGCOMM '92*, Aug. 1992.
[27] J. Postel, "DoD Standard Transmission Control Protocol," Network Information Center RFC-793, SRI International, Sept. 1981.
[28] S. Casner and S. Deering, "First IETF Internet Audiocast," *ACM Comp. Commun. Rev.*, vol. 22, no. 3, July 1992.
[29] K. K. Ramakrishnan, D. M. Chiu, and R. Jain, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer — Part IV: A Selective Binary Feedback Scheme for General Topologies," DEC Tech. Rep. DEC-TR-510, 1987.
[30] C. Ozveren, R. Simcoe, and G. Varghese, "Reliable and Efficient Hop-by-Hop Flow Control," *Proc. SIGCOMM '94*, Aug. 1994, pp. 89–100.
[31] B. Lyles and A. Lin, "A Class-Y mechanism and preliminary simulations," T1S1.5/94-207, July 11–15, 1994, St. Louis, MO.

One principle underlying this new paradigm is that the network should enable users to achieve good service.

Biographies

CHRISTOPHER LEFELHOCZ received an M.S. degree in computer science from the Massachusetts Institute of Technology and a B.S. degree in computer engineering from Carnegie Mellon University. He currently works at Fore Systems in the software engineering division. Prior to joining Fore, he spent a summer working at Xerox PARC on congestion control issues in best effort service.

BRYAN LYLES [M '82] received his undergraduate degrees from the University of Virginia, Charlottesville, and Ph.D. degree from the Unviersity of Rochester, Rochester, NY, in 1972 and 1983, respectively. Since 1988, he has been a member of the research staff at Xerox Palo Alto Research Center, working in the area of ATM networks. He has been *a faculty member at the University of Linköping*, Linköping, Sweden.

SCOTT SHENKER is currently a principal scientist at the Xerox Palo Alto Research Center. He received his Sc.B. (1978) from Brown University, his Ph.D. (1983) in theoretical physics from the University of Chicago, and spent the 1983–4 academic year at Cornell University as a postdoctoral associate. His most recent computer science research focuses on the design of integrated services packet networks and the related issues of service models, scheduling algorithms, and reservation protocols. His recent economic research addresses incentive compatibility and fairness in various cost-sharing mechanisms. Besides computer networks and theoretical economics, his other research interests include chaos in nonlinear systems, critical phenomena, distributed algorithms, conservative garbage collection, and performance analysis.

LIXIA ZHANG is an associate professor of computer science at UCLA. She received her B.S. degree in physics from Heilongjiang University, China, in 1976 and her Ph.D. in computer science from the Massachusetts Institute of Technology in 1989. From 1989 to 1995 she was a member of the research staff at Xerox PARC. Zhang is an active member of the Internet Engineering Task Force (IETF) and is currently serving on the Internet Architecture Board (IAB). Her research interests include network architectures and protocols, protocol implementations, and performance analysis.

INFOCOM '96 to Feature Keynote Address by Penzias

INFOCOM '96, the fifteenth Annual Joint Conference of the IEEE Computer and Communications Societies, will take place March 24-28, 1996 in San Francisco, CA, USA. INFOCOM '96 covers networking technology, when and where it happens.

Early registration for INFOCOM '96 ends March 1st, 1996. Detailed registration information and the full technical program can be found at the INFOCOM '96 web site at URL:

http://www.research.att.com/~hgs/infocom96/

Alternatively, you can contact Roch Guerin (guerin@watson.ibm.com) or Henning Schulzrinne (schulzrinne@fokus.gmd.de) for general information on INFOCOM '96, and Mary-Kate Rada at the IEEE Computer Society (phone +1 202 371 0101 or fax +1 202 728 0884) for specific registration information.

The INFOCOM '96 technical program will address the latest issues debated in standards bodies, and discuss the most recent technical advances in 44 technical sessions, which will present advances in areas such as multimedia systems, wireless systems, quality of service support, optical networks, ATM switch design, ATM flow control, network management, performance and modeling techniques, and many more exciting topics.

Six tutorials will provide the most up-to-date information on wireless networking, image and video compression, optical networking, high-performance networks, modeling and control of broadband network, and multimedia conferencing over the Internet. Panel sessions involving well known technical experts will provide original perspectives and lively discussions on hot topics such as wireless multimedia networks, ATM interoperability, multimedia synchronization, control of high-speed networks, simulation, and the economics of the Internet.

The keynote address, entitled "Two Views of Next-Generation Networking," will be given by Dr. Arno Penzias, who is Vice President of Research at AT&T Bell Laboratories. In that position, he is responsible for a broad range of research programs in the physical, materials, information and communications sciences. Dr. Penzias began his scientific career in 1961 when he joined Bell Laboratories as a Member of Technical Staff. He conducted research in radio communication and took part in the pioneering Echo and Telstar communications satellite experiments. As a scientist, he is best known for his contributions to astrophysics, which earned him the Nobel Prize for Physics in 1978.

A sought-after speaker on emerging trends, he has written a number of articles on information technology, especially its impact on business and society. In a recent book entitled "*Harmony: Business, Technology and Life After Paperwork*," he charts the course of the information revolution and its likely impact on the future of our work environment. In his keynote address, Dr. Penzias will touch upon many of the core issues that will shape the future of networks and the telecommunications industry, as well as their impact on daily life.