Congestion Control Mechanisms and the Best Effort Service Model

Panos Gevros, Jon Crowcroft, Peter Kirstein, and Saleem Bhatti
University College London

Abstract

In the last few years there has been considerable research toward extending the Internet architecture to provide quality of service guarantees for the emerging real-time multimedia applications. QoS provision is a rather controversial endeavor. At one end of the spectrum there were proposals for reservations and per-flow state in the routers. These models did not flourish due to the network's heterogeneity, the complexity of the mechanisms involved, and scalability problems. At the other end, proposals advocating that an overprovisioned best effort network will solve all the problems are not quite convincing either. The authors believe that more control is clearly needed for protecting best effort service. An important requirement is to prevent congestion collapse, keep congestion levels low, and guarantee fairness. Appropriate control structures in a best effort service network could even be used for introducing differentiation. This could be achieved without sacrificing the best effort nature of the Internet or stressing its architecture beyond its limits and original design principles. In this article we revisit the best effort service model and the problem of congestion while focusing on the importance of cooperative resource sharing to the Internet's success, and review the congestion control principles and mechanisms which facilitate Internet resource sharing.

The Internet was simply designed for packet delivery. However recent developments such as commercialization and the diversity of application requirements make it obvious that a more *concrete definition of the type of service delivered to the user* is needed. This description of the service delivered by the network is called the service model and documents the commitments the network makes to the clients that request service. It describes a set of end-to-end services and it is up to the network to ensure that the services offered at each link along a path combine meaningfully to support the end-to-end service.

Traditionally, in the Internet all packets are treated the same without any discrimination or explicit delivery guarantees. This is known as the *best effort service model*; all the network promises is to exert its best effort to deliver the packets injected into it without committing to any quantitative performance (quality of service, QoS) bounds.[1] Users do not request permission before transmitting, and therefore perceived performance is determined not only by the network itself, but also from other users' offered load, resulting in a complete lack of isolation and protection. The best effort service model has no formal specification; rather, it is specified operationally; *packet delivery should be an expectation rather than an exception*. The traditional applications and protocols were flexible, adaptive, and robust enough to operate under a wide range of network conditions without requiring any particularly well-defined service.

[1] *The QoS bounds are usually any combination of the three following metrics: throughput, end-to-end delay, and packet loss ratio.*

The Problem of Congestion

Congestion is the *state of sustained network overload* where the demand for network resources is close to or exceeds capacity. Network resources, namely link bandwidth and buffer space in the routers, are both finite and in many cases still expensive. The Internet has suffered from the problem of congestion which is inherent in best effort datagram networks due to uncoordinated resource sharing. It is possible for several IP packets to arrive at the router simultaneously, needing to be forwarded on the same output link. Clearly, not all of them can be forwarded simultaneously; there must be a service order. In the interim buffer space must be provided as temporary storage for the packets still awaiting transmission.

Sources that transmit simultaneously can create a demand for network resources (arrival rate) higher than the network can handle at a certain link. The buffer space in the routers offers a first level of protection against an increase in traffic arrival rate. However, if the situation persists, the buffer space is exhausted and the router has to start dropping packets. Traditionally Internet routers have used the *first come first served* (FCFS) service order, typically implemented by a *first in first out* (FIFO) queue, and drop from the tail at buffer overflow as their queue management strategy.

The problem of congestion cannot be solved by introducing "infinite" buffer space inside the network; the queues would then grow without bound, and the end-to-end delay would increase. Moreover, when packet lifetime is finite, the packets coming out of the router would have timed out already and been retransmitted by the transport protocols [1].

0890-8044/01/$10.00 © 2001 IEEE

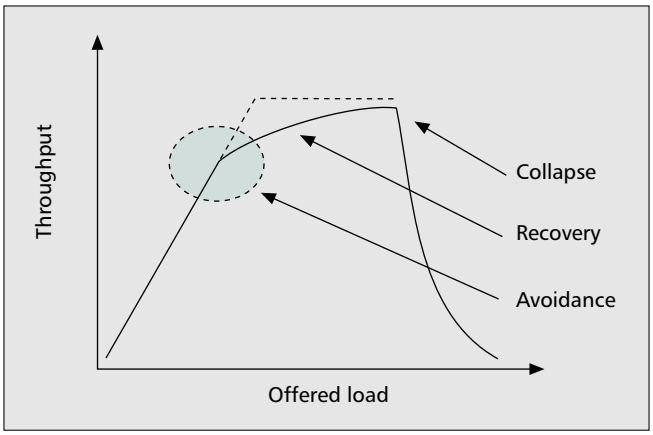

Figure 1. *Throughput as a function of the offered load.*

In fact, too much buffer space in the routers can be more harmful than too little, because the packets will have to be dropped only after they have consumed valuable network resources.

The Threat of Congestion Collapse

Congestion in the Internet can cause *high packet loss rates*, *increased delays*, and can even break the whole system by causing *congestion collapse* (or "Internet meltdown"). This is a state where any increase in the offered load leads to a decrease in the useful work done by the network (Fig. 1, the area beyond the "cliff"). The threat of congestion collapse is not a new one; it dates back to the early days of the Internet (then ARPANET) and can take several forms. In 1984 Nagle [2] reported on congestion collapse due to TCP connections unnecessarily retransmitting packets that were either in transit or already received at the receiver. This form of congestion collapse is a stable condition which results in throughput that is only a small fraction of the normal. This phenomenon, as predicted by Nagle, occurred several times in 1986–1987 with a large number of sites experiencing simultaneous slowdown of their network services for prolonged periods. At that time BBN, the firm maintaining the Internet backbone, responded to the collapse by providing additional link capacity. However, this could only ever be a temporary fix.

Another form of congestion collapse is the one from *undelivered packets*; in this case bandwidth is wasted by delivering packets that will be dropped before they reach their final destinations. Compared to the "classical" congestion collapse reported by Nagle, this form of congestion collapse is not a stable condition but one that can be reversed if the offered load is reduced. Other forms of congestion collapse reported by Floyd [3] include *fragmentation-based congestion collapse* in which the network transmits fragments of packets that will be discarded at the receiver since they cannot be reassembled into a valid packet, and *congestion collapse from stale packets* in which the network carries packets that are no longer wanted by the user (because the transfer took so long for instance).

TCP Congestion Control and the Role of Cooperation

In order to deal with congestion the Internet used end-to-end window-based flow control in its Transmission Control Protocol (TCP) [4], primarily for controlling demand on the receiver's bottleneck resources (memory and processing). Since 1987 TCP congestion control has been augmented with the Slow Start and Congestion Avoidance algorithms developed by Jacobson and Karels [5]; these algorithms became mandatory requirements for all Internet hosts [6]. The receiver-driven TCP flow control mechanisms have been the only congestion control methods available. This is why the term "flow control" is sometimes confused with "congestion control," although the former is only one method of the latter.

Few would argue that the TCP congestion control mechanisms have served the Internet remarkably well and formed the basis for its survival and success. The TCP window management algorithm uses the well proven principles of Additive Increase and Multiplicative Decrease (AIMD) [7]. AIMD manages to reconcile mutually contradictory objectives by being simple to implement, effective in the presence of congestion, efficient with respect to resource utilization, stable, scalable with the number of sources, and fair by providing equal shares to users sharing a scarce resource.

The Role of Cooperation

Nevertheless, it should be stressed that TCP congestion control owes its success to the fact that everyone was using it; in other words, to *cooperation*. So far it has been assumed that there is a *uniform response* to the congestion signals by all users. This assumption was indeed true because the Internet used to be a small network, operated by a technically knowledgeable community adhering to informal rules about congestion control and the use of Internet services. The operating systems in use were mainly UNIX and its variants, which allowed the "standard" congestion control algorithms of TCP to be deployed universally.

To this day users who misbehave (do not respond to the congestion signals as the "standard" TCP rules prescribe) capture more bandwidth than their fair share, seriously degrade the service delivered to cooperating users, and in general threaten the stability and operation of the entire system. There are only *guidelines* by the research community for cooperative congestion control behavior in the form of so-called *TCP friendliness*. Non-TCP flows are considered *TCP-friendly* if "their long-term throughput does not exceed the throughput of a conformant TCP under the same conditions" [3]. However, this definition is weak, since there are several TCP variants with widely different performance; moreover, the same average loss rate can affect throughput in different ways depending on the actual distribution of packet losses.

The Internet architecture has not incorporated any *incentives* for cooperative congestion control behavior. Furthermore, users do not have information about the behavior of other users against whom they are competing for network resources, so creating appropriate incentives is not an easy task. Misbehaving users are even implicitly rewarded by receiving a larger fraction of bandwidth than they would have received by being cooperative. The importance of detecting and penalizing misbehaving users was realized from the early days of the Internet. Request for Comments (RFC) 896 [2] suggests that a router detect and disconnect a misbehaving host, although it is acknowledged that such detection is a nontrivial task because the definition of a well behaved host in terms of its externally observed behavior is subtle. Floyd [3] points out that the incentives for cooperative user behavior can only come from the network itself, and therefore router mechanisms are an inescapable necessity.

It is well known from game theory that noncooperative actions may lead to suboptimal outcomes; this situation is often referred to as the *prisoner's dilemma*. Axelrod [8] discusses the necessary conditions for maintaining cooperation in a system as a stable state. Cooperative behavior leads to fair allocations of resources; however, it is necessary that fairness be precisely defined before creating any incentives for cooperation.

Fairness

The notion of fairness is of major importance in the best effort Internet due to the lack of explicit admission control and quantitative service assurances. Fairness is conceptually related to congestion control; under conditions of low load everybody's demands are satisfied; there is no need for trade-offs and no considerations for decisions that lead to fair allocation of resources. Fairness becomes an issue only when there are *unsatisfied demands* and users have to compete for their share. In an environment of competitive individualist users, the critical factor of cooperation relies on the underlying notion of fairness as well as incentives for adopting certain behaviors. The importance of fairness is not drawn out of thin air, but is a result of an *optimization-under-uncertainty* argument. Rational individuals who have to make decisions under uncertainty (having limited knowledge of their fate) tend to adopt what is called the *max-min* rule: "the greatest benefit for the least advantaged."

Although several definitions of fairness arise from various disciplines, in the networking world the most popular notion is indeed that of *max-min fairness* (also referred to as the classical notion of fairness). Max-min fairness is informally defined as "*each user's throughput is at least as large as that of all other users which have the same bottleneck*" [9]. Fairness can be examined macroscopically along a path (global view) or on a per-link basis (local view). To translate from a locally fair allocation decision to a globally fair one, each user (flow) should limit its resource usage to the smallest locally fair allocation along its path; this is known to result in a globally fair allocation [10].

Formally, let I be a set of users and $x = (x_i; x_i > 0 \text{ and } i \in I)$ the vector of the allocations to each user. The vector is called feasible if the sum of the allocations does not exceed the capacity of the resource.

Max-min fairness is a widely used technique for resource allocation in cases where some users' demand is smaller than others. It has been considered desirable in the networking community — in both the Internet Engineering Task Force (IETF) and the ATM Forum (for asynchronous transfer mode) — and operates as follows:

- Resources are allocated in increasing order of demand.
- A user is never allocated a share higher than its demand.
- All users with unsatisfied demands are allocated equal shares.

Initially all users get at least as much as the "small" user demands, and the remaining resources are *evenly* distributed among the users with unsatisfied demands. It follows that from those users with unsatisfied demands no one can increase its share without decreasing the share of a user with an already small one. This can be formally expressed as follows: a vector of allocations **x** is max-min fair if for any other feasible vector **y** there exists a user j such that $y_j > x_j$ implies that there exists user i such that $y_i < x_i < x_j$.

The Role of Policy

Fairness should not necessarily imply equal distribution of resources to all those users with unsatisfied demands. A fair allocation of resources is usually defined with respect to a given policy. Policy is the unified regulation of access to network resources and services based on administrative criteria. It can be expressed at different levels:

- Macroscopically at the network level taking into account topology, connectivity, end-to-end performance objectives and the dynamic state of the network
- At the node level where a set of mechanisms like classification, policing, buffer management, and scheduling allow administrative intentions to be translated into differential packet treatment [11]

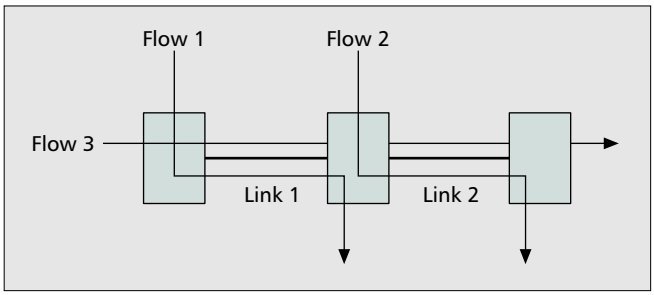

Figure 2. *Three flows sharing two links of the same capacity.*

Under a certain policy it may be justifiable (under certain criteria) to allocate more resources to some users than to others; this leads to a generalization of max-min fairness.

Weighted max-min fairness generalizes the concept of max-min fairness for the case where users have different rights to resource allocation. Each user i is associated with a weight w_i which reflects its right to a relative resource share. The weighted max-min fair allocations are calculated as follows:

- Resources are allocated in increasing order of demand *normalized by weight*.
- A user is never allocated a share higher than its demand.
- Users with unsatisfied demands are allocated shares *in proportion to their weights*.

The formal definition is similar to the one used for max-min fairness, but the allocations are replaced by the ratio x_i/w_i.

It has been argued that max-min fairness can be suboptimal in several contexts depending on the actual utility functions of the flows. Using logarithmic utility functions Kelly introduced the notion of *proportional fairness* [12] as a more suitable fairness model for bandwidth sharing. Proportional fairness tends to favor "short" flows over "longer" ones and has ties to the Nash bargaining scheme, which is known to be the only arbitration scheme to satisfy certain axioms from the economic theory viewpoint (such as Pareto optimality).

Proportional Fairness

Formally, a vector **x** is proportionally fair if it is feasible and if for any other feasible vector **y** the *aggregate of proportional changes* is zero or negative:

$$\sum_{i \in I} \frac{y_i - x_i}{x_i} \leq 0.$$

In a similar fashion, *weighted proportional fairness* generalizes the notion of proportional fairness for the case where user allocations are influenced by the *price per unit share* a user is prepared to pay (w_i). Then the feasible vector of allocations **x** is weighted proportionally fair if for any other allocations vector **y** the weighted sum of the proportional changes is zero or negative:

$$\sum_{i \in I} w_i \frac{y_i - x_i}{x_i} \leq 0.$$

The allocations resulting from the two types of fairness described above are illustrated with an example. Figure 2 shows three flows sharing two links of unit capacity. Let the notation for the allocations to flows 1, 2, and 3 be (x_1, x_2, x_3); then the max-min fair allocations would be (1/2, 1/2, 1/2) while the proportionally fair allocations would be (2/3, 2/3, 1/3). Assuming logarithmic utility function for each flow, proportional fairness requires that the "longer" flow 3 sacrifice its own utility (thus receiving a smaller share) for a greater sum of the utilities of all flows. It is known that TCP is biased toward "shorter" flows; flows that traverse a smaller number of links (because they are exposed to potentially fewer losses) or have smaller round-trip time (because they

update their window faster). However, the fact that proportional fairness results in allocations that favor such "short" flows, does not necessarily mean that TCP is proportionally fair.

Clearly max-min and proportional fairness are the same in the case of a single resource. Proportional fair shares depend on the number as well as capacity of the resources (links) on which a user places demand. The problem with this approach is that the end user is usually unaware of how many or how congested are the links the traffic goes through. The user tends to view the network as a single resource, unaware of the actual cost implications different actions might have. These costs are reflected in the prices the user is asked to pay, but again it is hard for the endpoints to infer and the network to calculate these prices and communicate them to the endpoints.

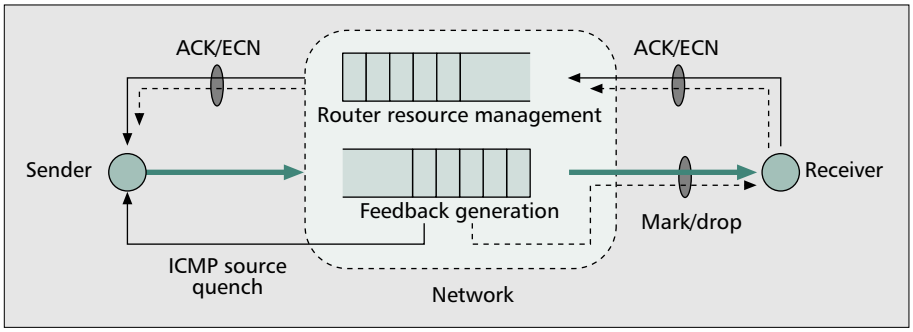

Figure 3. *Feedback flow control.*

Timescales of Bandwidth Sharing

In the Internet bandwidth is shared between many different flows; therefore, a decrease in resource usage by one flow may potentially increase the fair allocation of other flows which share part of their path with that flow, and vice versa. Of course the flows can increase/decrease their resource usage (by adjusting their sending rate or window) only after a certain propagation delay which depends on their flow control mechanisms and the feedback received by the network. It is possible that by the time a source adapts to an allocation which was known to be fair in the near past, traffic conditions (demand on resources) may have changed dramatically. This introduces oscillating behavior and makes important the issue of determining the appropriate timescales in which fairness should be examined in such an environment. Another issue related to fairness is the appropriate granularity at which to define a user. Even if protocol implementations are well behaved, applications may use them in a socially suboptimal manner. For instance, they can open many connections to the same destination, as has been the case in the past with some popular Web browsers. The granularity of a "user" for fairness and congestion control purposes is a policy issue which has not been addressed in the IETF. However, the general consensus seems to be toward that of a source/destination host pair being the most appropriate grouping for the definition of a network user from the router perspective.

Congestion Control Mechanisms

The Internet is decentralized by nature, comprising many heterogeneous administrative domains; therefore, resource management naturally involves both end-to-end as well as local (per-link) decisions. We identify two broad classes of congestion control mechanisms with regard to where these mechanisms are implemented: *host-based* and *router-based* mechanisms.

The entire Internet architecture was founded on the concept that all flow-related state should be kept on the hosts [13]; therefore, the congestion control mechanisms were mainly implemented in the end hosts. Upon detection of congestion the sources should inject their packets into the network more slowly. This mechanism is called *end-to-end flow control*.[2] In order for a host to be able to detect congestion, the routers must be able to provide the information that the network is currently (or is about to become) overloaded; this mechanism is called *feedback*. Flow control and feedback are conceptually related, so they are often referred to as *feedback flow control*. Although flow control should be aware of the feedback semantics, the exact mechanisms used to implement either are orthogonal, so we decided to treat them separately in our context. The feedback mechanism is distributed and can be implemented partly or entirely at the end hosts (receiver side) or routers. Packet drops were, and to a great extent are still, the only means for a router to fight congestion. The sources become aware of the packet drops, interpret them as a congestion indication, and reduce their rates.

The feedback from the network and the response from the source are the foundations of Internet congestion control and are very important because they facilitate decentralized resource allocation. However, with decisions made at the end hosts and treatment of the network as a black box that simply drops packets, there is clearly a limit on how much control can be achieved over the allocation of network resources. This also limits the range of services the network is capable of offering.

Routers, on the other hand, know exactly how congested they are and can therefore perform more drastic resource management. Thus, the introduction of router mechanisms for congestion control that will enable the network to more actively manage its own resources seems inescapable [3]. These mechanisms can be used as building blocks for providing

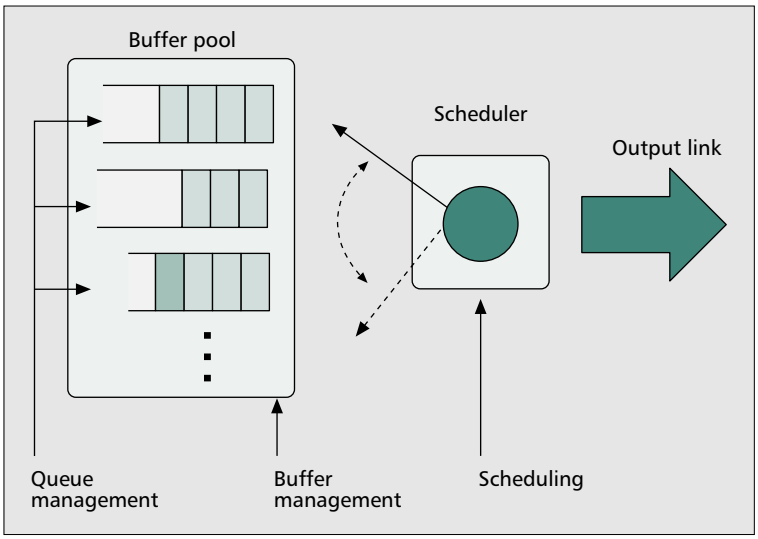

Figure 4. *Router resource management actions.*

[2] *Flow control has also been used for controlling demand on the receiver's bottleneck resources (memory and processing power), which is irrelevant to network congestion.*

higher-level resource management mechanisms such as *link sharing*[3], *penalty boxes*,[4] and *pricing*, which by means of financial incentives controls the sharing of network resources.

The extension of router functionality per se does not contradict the design philosophy of the Internet where all state should be kept at the end hosts or, better, at the edges of the network. Routers have two conceptually orthogonal methods of managing their own resources: *scheduling* to directly manage bandwidth allocation on an output link, and *queue/buffer management* to manage buffer space and queue occupancy, respectively, and thus indirectly affecting bandwidth allocation.

Congestion Control Phases

Clearly congestion can be avoided at the expense of low resource utilization; however, this is usually undesirable. Thus, the goal of any congestion control mechanism, with respect to resource utilization, is to operate the resource (link) in a region close to its capacity (see the circled area, called the "knee," in Fig. 1). There are two phases in congestion control:

- *Congestion avoidance* when the system operates about the knee
- *Congestion recovery* (often confusingly referred to as *congestion control* in the literature) when the state of the system is between the knee and the cliff, and congestion has occurred so that the total load should be decreased to avoid collapse (from which it might not be possible to recover)

We next treat in turn each of the four classes of congestion control mechanisms identified above.

End-to-End Flow Control

In control theory a controller changes its input to a black box and observes the corresponding output. The goal is to choose the input as a function of the observed output so that the system state conforms to some desired objective, provided that the system state can be observed. From a control-theoretic viewpoint the end host flow adjustment is the response to a servo-control loop which needs to match the source's sending rate to the rate that corresponds to its fair share at the bottleneck link. The problem is that the appropriate bottleneck service rate becomes known to the source after a delay, and the new rate (after any adjustments) takes effect at the bottleneck only after another delay. The precision of the servo-control loop determines performance; if the queue at the bottleneck link is empty, throughput will be less than the maximum. If there are always packets in the queue, the link will never be idle, but if the queue size grows beyond a limit, packets will start being discarded. However, in flow control, the output of the system (the rate of a flow as seen at the receiver) does not depend only on the actions of that particular flow, but also on the actions of all other flows sharing the same path.

Other flow control issues involve the *decision function* (how the feedback information is interpreted), *frequency of control* (how often the source decides to adjust its window/rate), and *control function* (how the window/rate is adjusted). The above issues depend on the end-to-end network path properties (available bandwidth, delay, loss) as well as the nature of the feedback signal, which has to be predictable and well defined; otherwise, end-host adjustments cannot be used to implement resource allocation policies or even to control loss.

Except for the primary goal of rate matching, a flow control mechanism tries to achieve certain, sometimes mutually contradictory, objectives that allow interesting design trade-offs and lead to a wide range of mechanisms. Flow control schemes generally fall in two broad categories: *open loop* and *closed loop*.

Open-loop flow control is acceptable only in an environment without considerations about the impact of individual actions to other network users. In an open-loop flow control scheme the sender *describes its rate* to the network with parameters like *burst size* and *interburst interval*. Simply stating the rate is not sufficient because b packets/s may be 1 packet every $1/b$ s, but it can also mean a burst of b back-to-back packets every second, which might be unacceptable for a gateway that does not have enough buffer space to store the burst. The network examines the parameters given by the sender and if the request can be granted (admission control based on availability or policy criteria) it reserves resources, corresponding to these parameters, along the path from the sender to the receiver. The sender simply ensures that its rate conforms to the given description, and in this fashion network congestion is avoided. This paradigm fits nicely in a connection-oriented architecture like IntServ but cannot be enforced only with end-to-end mechanisms; it requires resource management mechanisms in all the routers.

The difficulty with open-loop flow control is to accurately describe source behavior using a small set of parameters since the network must be aware of these parameters for admission control calculations. However, sources may be bursty and delay-intolerant; reservations at the peak rate do not usually lead to the most efficient use of bandwidth, preventing statistical multiplexing gains. It is therefore useful for the source output to be smooth. Other options for source description parameters include *average rate* or the use of a *linear bounded arrival process* (LBAP) [14]. An LBAP-constrained source bounds the number of bits transmitted in a time interval by a linear function of time. An LBAP can be used to describe a source with a known long-term average sending rate which can occasionally deviate from that average and transmit in bursts of a known maximum size. A *leaky bucket regulator* [15] is a mechanism for regulating the size of bursts allowed to a source characterized by LBAP. Intuitively the regulator collects tokens in a bucket and sends a packet only if the bucket has enough tokens otherwise the packet waits until enough tokens have been accumulated in the bucket or until it is discarded. The effect of a leaky bucket is to limit the size of bursts to a little more than the buckets depth (since tokens can arrive while packets are being transmitted).

Closed-loop flow control schemes target more dynamic network environments where it is a requirement for the sources to dynamically adapt their rate to match their fair share of network resources. The fair share usually fluctuates, and the sender must be able to track these changes and adjust its rate to allow for more efficient resource utilization. Closed-loop schemes can be *adaptive window*, in which the source indirectly controls the transmission rate by modifying the number of packets sent but not yet acknowledged (window), or *adaptive rate*, in which the source, every time it sends a packet, sets a timer with a timeout value equal to the inverse of the appropriate transmission rate and transmits the next packet when the timer expires. The potential damage to the network is constrained in different ways, but win-

[3] *Link sharing allows routers to control the distribution of bandwidth on a link based on local policies which allow multiple organizations to have access to a guaranteed share of the link bandwidth during congestion and optionally make unused bandwidth available to other organizations sharing the link.*

[4] *The purpose of a penalty box is to detect and restrict unresponsive and high-bandwidth flows in times of congestion, thus creating incentives for the use of end-to-end congestion control procedures.*

dow-based schemes are easier to implement because they do not require a fine-grained timer, which is hard to implement in non-real-time operating systems. If a closed-loop flow control scheme appears ineffective, either the sources suffer from excessive packet loss or the network resources are underutilized.

Feedback Mechanisms

The mechanism used for notifying the sender about network congestion or the appropriate sending rate is called the feedback, and inherently involves both the routers that generate the congestion signals and the receiver host(s) that propagates the signal to the sender for interpreting it accordingly. Closed-loop flow control mechanisms and overall network performance rely heavily on feedback. Without a feedback mechanism a source would be clueless as to what to do with its sending rate, and the network could become unstable, unfair, and either congested or underutilized. Feedback involves information about the state of the system, so in principle it should originate from the network and ultimately be delivered to the sender. The sender receives feedback either directly from the network or from the network *via the receiver*; therefore, there are two forms of feedback: *implicit* or *explicit*.

Implicit Feedback

Implicit feedback requires the end-hosts to be responsible for monitoring the performance of their own transmissions (delay, loss) for indications that will let them infer the state of the network and determine their appropriate sending rate. Nevertheless, it is debatable how accurately this can be derived. The most common form of implicit feedback signal is *packet drop* and has been traditionally used by Internet routers. However, packet drop is not necessarily an indication of congestion, for instance in error prone wireless links. Another proposed method of implicit feedback is the observation of the rate at which packets emerge from the bottleneck [16] or the measurement of the change in end-to-end delay as the transmission rate changes [17].

The advantage of implicit feedback is simplicity in the routers; routers are left to focus only on resource allocation, and do not have to calculate and produce an appropriate feedback signal. However, the scheduling mechanisms must be known to the end hosts for implicit feedback to be useful; otherwise, the observed performance may be misleading and not accurately describe the actual congestion state of the network. For example, with FIFO scheduling an increase in the rate may lead to an increase in the observed throughput, although queues may have already started building up and the total delay has increased.

Explicit Feedback

In principle explicit feedback can be in the form of *congestion notification* or *rate indication*. Due to the limitations in the information that can be carried in protocol headers explicit feedback can be *binary* (in its lowest granularity : "congestion experienced") or *multivalued* (usually limited to a small number of values: "how much congestion has been experienced"). In the case of binary feedback the appropriate operating point is found through an iteration process of network feedback and host adjustments. For explicit feedback the only methods proposed for TCP/IP networks is the ICMP Source Quench messages and Explicit Congestion Notification (ECN) proposal [18] (the idea first appeared in the DECbit scheme [19]).

The ICMP Source Quench message is sent by the IP layer of a host or router to throttle back a sender in case the host/router runs out of buffers or throws datagrams away [20]. ICMP Source Quench is very rarely used in the Internet, and although there is no substantial evidence, the current feeling is to deprecate this message because it consumes bandwidth at times of congestion, and is generally an ineffective and unfair fix to congestion [21, 22].

In the ECN feedback scheme the router sets a bit in the packet header (CE bit) whenever it detects incipient congestion. The receiver copies this bit into the header of the acknowledgment packet, and the flow control mechanism at the sender is responsible for adjusting the window (or rate) based on a certain algorithm. The algorithms used for congestion detection and window adjustment as responses to explicit feedback are part of the queue management and flow control mechanisms, respectively. Explicit feedback implies an extra mechanism in the router, but on the other hand provides more quantitative control information which can be valuable for the adjustment process. Explicit rate indication [23] is another method of explicit feedback in which the switches perform rate allocation and the calculated rates are explicitly communicated back to the sources (via the receiver) as information in the packet headers; it has been used in ATM networks but not in the Internet.

Scheduling Mechanisms

Scheduling determines the service order of the packets and therefore is the most direct control over how a network serves its users. There have also been claims that scheduling is the only effective means of resource management and that queue management alone is not sufficient for controlling bandwidth usage, but this needs further investigation. The scheduling discipline controls the bandwidth allocation by serving a certain number of packets from each flow in a given time interval.

There are several factors that have to be considered in the design of a scheduling discipline. The most important design factor is the number of *priority levels* that may be realized as separate queues. Assuming n priority levels and the higher-numbered levels corresponding to higher-priority users, the scheduler only serves a packet from priority level k if there are no packets waiting to be served at higher priority levels k, $k + 1 \ldots, n$ (multilevel priority with exhaustive service). The *service order within a priority level* must also be defined, especially when the priority level serves aggregates of flows with different delay requirements [24]. This can be something as simple as FCFS or more complex by sorting packets on the basis of service tags calculated by the scheduler. Another design decision is the degree of *flow aggregation within a priority level*; each flow may be treated separately (per-flow scheduling), or several flows may be treated at the same priority level (class). However, the issue of resource management for flows that belong to the same class still remains, and it is not guaranteed that all flows in the same class will receive the same QoS. The service order within the class and the cooperation between the flows are important here.

Last but not least is the issue of *work-conserving* vs. *non-work-conserving* scheduling disciplines. A work-conserving scheduler is idle only when there are no packets that require service. In contrast, a non-work-conserving scheduler may be idle even if there are packets awaiting service. The reason for doing so is that it serves packets only when these become eligible. Eligibility times can be computed in such a fashion that the packet stream after the scheduler conforms to criteria related to packet interarrival time variation (jitter) and burst size, which is crucial to the buffer provisioning of the downstream routers. A scheduling discipline must be easy to implement, provide fairness and protection, be able to satisfy certain performance bounds (deterministic or statistical), and

have efficient admission control procedures. Sometimes these requirements can be contradictory and lead to trade-offs.

The simplest scheduling algorithm is FCFS, implemented with a FIFO queue, which serves packets in order of arrival. With FCFS all users experience the same average delay even if only a small number of them are responsible for the overload (in other words, greed is rewarded). So FCFS does not protect users and can be unfair too, since it distributes link bandwidth according to queue occupancy distribution. With FCFS the way buffer space is managed has a direct impact on the way bandwidth is managed.

FCFS cannot provide max-min fair allocation, this can be achieved by an ideal work-conserving discipline called *generalized processor sharing* (*GPS*) [25]. GPS serves packets as if they were in separate logical queues, by visiting each nonempty queue in turn and serving an infinitesimally small amount of data from each queue; in any finite time interval, it can visit every logical queue at least once, skipping potentially empty queues. The queues can have weights associated with them and receive service in proportion to their weight, in which case GPS achieves *max-min weighted fair* sharing.

GPS is only a model and cannot be implemented in practice. The simplest emulation of GPS is *round-robin*, which serves a packet from each nonempty queue instead of an "infinitesimal amount of data." When the queues have weights associated with them they get served in proportion to their weights; the scheme is then called *weighted round-robin* (*WRR*). In order to allocate bandwidth fairly, WRR requires knowledge of the average packet size for each queue but this is not realistic given the characteristics of the sources. Moreover WRR is fair only when examined in time-scales larger than the round-time (the time taken to serve each queue once).

Weighted tair queuing (*WFQ*), [26] is an approximation of GPS that does not require knowledge of the average packet size for each queue. Instead, it emulates GPS by associating each packet with a finish number which corresponds to the time this packet would complete service had it been served by GPS. The packets are then served in order of these finish times and WFQ can provide a flow with QoS guarantees. For leaky bucket constrained sources and arbitrary topology networks of GPS servers, Parekh and Gallager [27] proved that there is a bound on the worst case end-to-end delay experienced by a flow that passes through a series of GPS schedulers. The bounds for GPS networks apply to networks of packet GPS (synonym to WFQ) schedulers for small packet sizes. The original WFQ is computationally expensive to implement so several variations of WFQ have been proposed that are optimized for software and/or hardware implementation such as worst-case WFQ (W2FQ)[28], self-clocked fair queuing [29], and deficit round-robin [30].

Scheduling disciplines are a very important part of higher-level resource management mechanisms, like link sharing [31], which are expected to play a key role as building blocks of future IP service models. Although, strictly speaking, fair queuing may not be absolutely necessary for implementing a certain QoS policy (bandwidth allocation, delay, or jitter guarantee), it is clear that some form of active resource management is required that can provide protection and enforce bandwidth allocation policies.

Buffer and Queue Management Mechanisms

Scheduling cannot by itself offer a solution to resource management inside the network, mainly because traffic can arrive in bursts. So unless there is enough buffer space to absorb these packet bursts and transmit them in subsequent silence intervals, the loss rate can be very high irrespective of the scheduler; thus, buffering is essential. However, buffer space in the router is finite and can be exhausted when the traffic arrival rate exceeds link bandwidth for a sufficiently long time. Even if the buffer space were infinite, it (surprisingly) would not solve the problem. Nagle [1] observes that a datagram network with infinite storage, FCFS scheduling, and finite packet lifetimes (based on the time to life, TTL, field in the IP header) under overload conditions will drop all packets.

Buffer Management

The role of buffer management is to determine how the buffer space is shared between the different flows that traverse the gateway and, in particular, those flows that use the same output interface. There is a wide variety of possible strategies for buffer allocation, static or dynamic and based on different criteria, number of flows, current or past bandwidth allocation, and buffer occupancy. The two most popular buffer management schemes are *shared buffer pool* and *per-flow allocation*. There are more sophisticated ways of allocating buffer space, mostly influenced by router design and implementation issues, and they are probably the least well understood aspects of congestion control today. In a shared buffer pool buffers are used on a first come first use basis, and there is clearly no protection between the flows since one flow can occupy all the buffers and starve all other flows by simply sending fast enough. Due to its simplicity and implementation efficiency this scheme is found in most Internet routers today.

Per-flow allocation protects flows from each other by keeping track of buffer utilization and dropping packets based on the buffer occupancy level of each flow. This is considered expensive and cannot scale in terms of processing power to meet the requirements of large numbers of flows in the backbone routers.

The larger the maximum allowed queue length, the larger the size of burst that can be absorbed without dropping packets. However, it is obvious that long queues in the routers increase end-to-end delay. Delay is a very important performance measure for many applications therefore maintaining in steady state short queues results in higher throughput and lower end-to-end delay while a high maximum queue limit can be useful for absorbing occasional large bursts of packets.

Queue Management

The role of queue management is to control the length of the queue and potentially which flows occupy it, by selecting which packets to drop and determining when this is appropriate. Queue management mechanisms are orthogonal and complementary to both scheduling (which determines the service order) and buffer management (which determines the number of queues per output interface; aggregate/class based or per-flow queues). However, it should be noted that packet drop has traditionally been interpreted by TCP as a congestion indication; thus, the queue management mechanism is also the place where the feedback to the source originates by either dropping (or marking) packets. In a congestion situation where the offered load persistently exceeds link bandwidth, and the sources do not react uniformly to congestion the queue management discipline has a stronger impact on bandwidth sharing than the scheduling discipline itself [32]. There are generally two options for managing a queue at a router that correspond roughly to congestion recovery and congestion avoidance as we discuss below.

Queue Management for Congestion Recovery — Traditionally Internet routers have managed the queues at their links (almost always one queue per output interface) by setting a maximum length for each queue, accepting packets until the

maximum length is reached and then dropping subsequent incoming packets until space becomes available in the queue. This method is known as *tail drop* because packets arriving at the end of the queue (the tail) are dropped when the queue is full. Tail drop has served the Internet well for years, but has two serious disadvantages: it sustains full queues, and it can cause lockout due to traffic phase effects; we describe these two problems in turn. Tail drop by nature signals congestion via packet drops very late (only when the queue overflows) and this allows the queue to maintain a full (or almost full) status most of the time. With tail-drop routers it is also possible to introduce global synchronization in the network because when a queue overflows packets from several sources are often dropped and all these sources reduce their transmission rate simultaneously and their control actions become synchronized. This can lead to reduced link utilization (if there are intervals when the queue is empty) or lockout phenomena (flow segregation) where a few sources monopolize queue space, preventing others from getting in [33].

Discrimination against some of the flows is exacerbated by the fact that traffic sources in datagram networks can be periodic in nature (from flow control protocols based on round-trip times like TCP, to real-time audio and video applications). Control theory suggests that periodicity can have strong nonlinear interaction with deterministic control mechanisms in the gateways.

Besides tail drop, another deterministic queue management mechanism that gets triggered on buffer overflow is *drop-from-front*. The gateway drops the packet from the front of the queue upon arrival of a new packet. Lakshman *et al.* [34] have shown that this method improves the performance of TCP by allowing the congestion indication signal to reach the sender faster than waiting for the full queue to be transmitted first. It also prevents lockout and improves fairness because dropping a packet from the front of the queue leads to a loss distribution that follows the buffer occupancy among connections at the instances when the queue is full.

It has been realized that global synchronization and lockout phenomena can be avoided by introducing randomization in the network. One such discipline is the *random drop* where the gateway randomly selects a packet to drop from the queue when a new packet arrives at a full queue. The intention of this discipline is to notify those users whose traffic contributes more to the congestion of the router. The intuition behind this is that a packet randomly and uniformly selected from all incoming traffic will belong to a particular flow with a probability proportional to the bandwidth share of that flow on that gateway. Random drop gateways are reported [35] to achieve improved fairness for late-starting connections and slightly improved throughput for connections with longer RTTs. However, the operation of randomly selecting a packet to drop can be computationally expensive.

Active Queue Management for Congestion Avoidance — The last two strategies described above solve the problem of lockout but do nothing to solve the problem of full queues. This happens because they do not react to congestion fast enough but only after congestion has occurred. In other words, they are congestion recovery mechanisms rather than congestion avoidance.

The solution to the full queues problem can come only if routers take measures to prevent congestion before it happens, and this can be achieved by dropping packets proactively rather than reactively. This proactive approach is called *active queue management* and allows routers to control which packets to drop and when this should happen in order to avoid congestion. Moreover, this approach need not necessarily use packet drops (the traditional method of congestion notification) but can also mark a packet and notify the source to reduce its rate. The mark can consist of setting a bit in the packet header or some other method understood by the transport protocol.

The primary goal of active queue management is congestion avoidance, but there are other goals: avoid global synchronization, eliminate the bias against bursty traffic, maintain upper bounds on router queue sizes even in the presence of noncooperating flows, penalize aggressive flows, reduce the number of packet drops, and provide lower-delay interactive service [36]. It was also hoped that it would be able to eliminate the bias against long RTT connections, although this bias proved to be an artifact of the TCP window increase and decrease algorithm rather than due to the gateway queue management discipline.

Random drop, although originally conceived as a congestion recovery mechanism, is also proposed for congestion avoidance by initiating the dropping of packets when congestion is anticipated instead of only when the queue becomes full. This enhanced version is called *early random drop* (*ERD*) The intention is that the drop rate (rate at which randomly and uniformly selected packets get dropped) is derived from the level of congestion at the gateway which is inferred from the number of packet arrivals between drops. A requirement for any congestion avoidance scheme is congestion detection. This was implemented in the simplest form by a static threshold in queue length, although more elaborate measures like exponentially weighted moving averages or link utilization have been proposed. The control interval is chosen in terms of packet arrivals, and the drop probability is fixed. An appropriate number of random variables are drawn in the packet range corresponding to the control interval, and these random numbers (uniformly distributed) are used to drop packets as they arrive by counting. ERD gateways are certainly an improvement over drop tail because they alleviate to a great extent the problem of flow segregation. However, they are not sufficient for providing fair bandwidth allocation and cannot successfully contain aggressive sources. ERD gateways are also biased against bursty traffic (like drop tail); this bias occurs because the contents of the queue do not necessarily reflect the average traffic.

Given the shortcomings of ERD gateways, a new mechanism was proposed by Floyd and Jacobson called *random early detection* (*RED*) [38]. RED gateways attempt to mark packets sufficiently frequently to control the average queue size and avoid the biases described above. The mechanism works as follows. The gateway detects incipient congestion by computing the average queue size; when this exceeds a preset threshold, arriving packets are marked (or dropped) with a certain probability that is a function of the average queue size. The average queue size is kept low, but occasional bursts can pass through unharmed. During congestion the probability of marking a packet from a particular flow is roughly proportional to the bandwidth share of that flow. There are two distinct algorithms operating in a RED gateway:

- The average queue size computation, which uses a low-pass filter with an exponentially weighted moving average, which determines the degree of burstiness that will be allowed
- The algorithm for calculating the packet marking probability, which determines how frequently packets get marked given the current level of congestion

RED is the most prominent and widely studied active queue management mechanism and successfully addresses the problems found in its predecessors. However, it is very difficult to parameterize RED in order to perform well under different traffic conditions. In almost all studies the parameter settings are based on heuristics, and the proposed configuration is suitable only for the particular traffic conditions studied. It is possible that the performance of a RED gateway to approach that

of a drop tail gateway for a given set of configuration parameters and traffic conditions. Feng [39] proposed a self-configuring (adaptive) active queue management mechanism which effectively reduces packet loss rates and maintains high network utilization across a wide range of traffic conditions.

Selecting which packets will be dropped is a very powerful mechanism that enables the gateway to implement specific resource management policies, making active queue management a key mechanism for the differentiated services Internet as proposed by Clark [40]. This is also backed by the widespread belief that mechanisms like WFQ scheduling or any form of per-flow queuing does not scale with the number of flows, the bandwidth of the links, and the processing power and memory requirements of the routers in the core of the network, although this requires further investigation.

Directions for Internet Service and the Role of Control Mechanisms

Ideally it would be desirable to have a single network which offers a number of end-to-end service options that exactly match individual application requirements at any given time without any perceived deterioration in QoS. However, given the diversity of application requirements and current experiences, such a network might not be possible or in fact even required. It is known that even reservation-capable networks which offer guaranteed QoS can have high blocking rates if the demand grows beyond certain limits for a certain level of provisioning. Providing QoS guarantees in the Internet has been the holy grail of modern networking. Indeed, extending the traditional best effort service model has proven to be not only an extremely complex task but quite controversial too.

The IETF Efforts

In recent years the IETF undertook two major efforts in order to enhance the services offered by the Internet. The first attempt was to develop a new service model called the integrated services architecture (IntServ) [41] which provides end-to-end QoS capabilities on a per-flow basis. The model involves two traffic classes:

- The best effort class for applications that are able to adapt to whatever QoS is available from the network
- The guaranteed service class for real-time applications which require "hard" QoS guarantees in order to be usable

The IntServ model suffers from scalability and manageability problems, and it has not been widely adopted by the Internet service providers (ISPs), although several router vendors implemented IntServ in their products.

The most recent IETF attempt is the differentiated services architecture (DiffServ) [42] which standardizes the *per-hop behaviors* (PHBs) as functional blocks out of which end-to-end services can be built. DiffServ aims to provide statistical QoS guarantees for traffic aggregates, as opposed to individual flows, and makes the fundamental assumption that *the Internet will continue to be dominated by best effort traffic*.

Overprovisioning

It is well known that the service of a best effort network is as good as its provisioning. Indeed, one of the major reasons for Internet success has been the fact that the network has generally been well enough provisioned to avoid congestion growing at least as rapidly as demand. Thus, it has been argued that the main problem of the Internet is not a technical one but an economic one, since the existing IP technology, even without any complex congestion control mechanisms, can provide satisfactory service in the absence of congestion. This view is usually backed by arguments for advances in optical transmission technology that promise more and cheaper link capacity.

It is believed that long-term congestion is better solved by installing extra bandwidth at the bottlenecks. However, this is feasible only within a well specified scope of deployment in order to be a truly cost-effective alternative to complex QoS structures and dynamic resource management schemes that are good only for transient congestion. On a global scale, overprovisioning is considered an economically prohibitive luxury since bandwidth is very unlikely to become infinite and cheap in the near term [43]. Even if prices drop, there will always be bottlenecks where resource allocation will have to be regulated and congestion controlled to provide the appropriate levels of service to everyone. Indeed, in the past, similar forecasts based on expectations for increased availability in resources and drop in prices were proven wrong. More and cheaper memory, high-speed links, and fast processors did not alleviate congestion problems. On the contrary, it was precisely at the point when high-speed LAN technology was introduced that interest in congestion control techniques increased [44]. This was caused by the bandwidth mismatch between the fast LANs and the relatively low-speed wide-area links which created congestion at the interconnection points.

However, the arguments for reservations and guaranteed QoS vs. overprovisioning are the two extremes of the spectrum. Shenker [45] argues that the amount of bandwidth needed to offset the benefits of extending the service model depends in detail on the utility functions of applications and the service model being offered. Proper evaluation requires judgment of future developments, about which little is known and opinions vary widely. He also conjectures that there will be enough bandwidth savings by offering multiple service classes to outweigh the costs of deploying extra mechanisms.

Odlyzko [46] has proposed running parallel network infrastructures ("channels") for different classes of service with higher prices for using the better provisioned and therefore (allegedly) less congested classes. *Pricing* would be the primary congestion control tool for achieving differential QoS in a best effort network without the need for additional control mechanisms.

Game Theoretic Aspects of Internet Congestion

The authors believe that mere overprovisioning cannot be the solution to Internet congestion. Free public goods are known to shift toward a state where demand grows beyond supply (known as "the tragedy of the commons" paradigm in economics) resulting in bad performance for everyone and loss of the potential benefits that would result from a more "sensible" use of the common "good." Overprovisioning can certainly be helpful in the best effort context by making service deterioration due to congestion less frequent and easier to tolerate. However, the instinctive solution is to impose more control for protecting the best effort Internet from abuse and service degradation. There is already evidence that use of appropriate congestion control mechanisms, (e.g., active queue management) can significantly contribute to congestion avoidance, and ensure fairness, stability, and high utilization levels.

Historically the problem of congestion has been approached from different viewpoints; by applying queuing theory, control theory, and lately economic principles, game theory in particular. The game theoretic aspects of Internet congestion, although they have long been realized, started receiving increasing interest in the research community recently. Solutions to the tragedy of the commons problem can be either:

- *Cooperative*
- *Authoritarian*
- *Market solutions* [1]

Cooperative solutions (everyone agrees to be well behaved) are adequate for a small number of players where "cheating" on behalf of a player is usually obvious to the others, but they tend to break as the number of players increases, and therefore mutual agreement and monitoring becomes impractical. Authoritarian solutions are effective when the behavior of other players can be monitored but tend to fail when the definition of good behavior is subtle (this also applies to cooperative solutions). Market solutions are applicable in certain cases when the rules of the game can be changed (e.g., price changes) and are known to be quite effective in situations where the resources are valued differently by different users. This is a relatively new direction in dealing with Internet congestion and requires further investigation.

The Internet is a large complex system that involves interactions of various entities at different levels. The interactions occur not only between end users (agents or transport protocols) and service/content providers but also at a higher level between larger entities; network service providers (offering backbone services) and access providers (or ISPs). The actions of these entities affect network conditions and the resulting resource allocations. Thus, it seems natural that the solution to the problem of uncoordinated access lies in applying different types of solutions to the different levels of the game.

The mechanisms described in this article can be applied to control the interactions, resource allocations, and ultimately congestion at different levels in this complex game. For instance, end-to-end flow control may be particularly relevant to end users, while a link sharing scheduling mechanism may be of great importance for resource management on an expensive link connecting two large backbone networks. Moreover, the mechanisms should provide appropriate knobs which encompass any of the above mentioned types of solutions. For example, in the case of link sharing, "borrowing" [31] may be allowed between agencies sharing the same link; that would be a cooperative type of solution. Knobs which allow distributed resource sharing and resolve congestion through "market-type" solutions have been gaining in importance lately [47].

Best Effort Service and Differentiation

Appropriate congestion control mechanisms can also be used to provide *relative differentiated services* [48] in a best effort context. Best effort was conceived with nondiscrimination in mind; however, *service differentiation and best effort service are conceptually orthogonal*. Thus, it is possible to create classes with qualitatively different QoS characteristics (proportional bandwidth, delay, or loss) and price them accordingly. The QoS spacing between the classes would be consistently proportional but there would be no quantitative assurances of any kind. It would be ultimately up to the end user to decide whether the price paid for using a class was fair given the QoS offered by the class at a given time. Introducing service differentiation in this fashion would be a step in the direction of "market-type" solutions to the problem of congestion.

Conclusions

Best effort service has been tremendously successful for data traffic, which today accounts for the vast majority of Internet traffic; there are no indications that this will stop being the case in the future. The main reason for pursuing QoS was concerns about the requirements of emerging real-time and streaming multimedia applications, which could not be met in the existing service model. Nevertheless, it has been amply demonstrated that many popular applications (packet audio, videoconferencing) are able to adapt to dynamic network conditions by changing their transmission rate using different coding techniques, and therefore perform sufficiently well under moderate congestion levels. Thus, it is likely for the Internet to evolve toward a best effort network which, if controlled and provisioned appropriately, will be able to satisfy the majority of popular applications that are willing to tolerate service deterioration due to transient congestion. This could sustain a large market for best effort service and would limit the applicability of service models for guaranteed QoS to corporate intranets and virtual private networks.

Most of the congestion control mechanisms presented in this article, the router-based ones in particular, were almost exclusively studied in the context of guaranteed QoS and real-time traffic. There has been considerably less research in their use within the best effort service framework, so there is a widespread misconception that the best effort service model necessarily implies simple FIFO queues in the routers. If appropriately used these mechanisms could provide, for instance, preferentially lower delays to "fragile" interactive applications (like telnet) without striving to provide any quantitative QoS guarantees. The authors believe that their use can considerably improve, enhance, and protect the best effort service model and that therefore this is a direction which deserves further investigation.

References

[1] J. Nagle, "On Packet Switches with Infinite Storage," *IEEE Trans. Commun.*, vol. 35, 1987, pp. 435–38.
[2] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC 896, IETF, Jan. 1984.
[3] S. Floyd and K Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Trans. Net.*, Aug. 1999.
[4] J. Postel, "Transmission Control Protocol," RFC 793, IETF, Sept. 1981.
[5] V. Jacobson, "Congestion Avoidance and Control," *ACM Comp. Commun. Rev.*, vol. 18, no. 4, Aug. 1988, pp. 314–29.
[6] R. T. Braden, "Requirements for Internet Hosts — Communication Layers," RFC 1122, IETF, Oct. 1989.
[7] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Comp. Networks and ISDN Sys.*, vol. 17, no. 1, 1989, pp. 1–14.
[8] R. Axelrod, *The Evolution of Cooperation*, HarperCollins, 1984.
[9] J. M. Jaffe, "Bottleneck Flow Control," *IEEE Trans. Commun.*, vol. 29, no. 7, July 1981, pp. 954–62.
[10] D. Bertsekas and R. Gallager, *Data Networks*, Englewood Cliffs, NJ: Prentice Hall, 1991.
[11] R. Rajan *et al.*, "A Policy Framework for Integrated and Differentiated Services in the Internet," *IEEE Net.*, vol. 13, no. 5, Sept. 1999, pp. 36–41.
[12] F. P. Kelly, "Charging and Rate Control for Elastic Traffic," *Euro. Trans. Telecommun.*, vol. 8, 1997, pp. 33–37.
[13] D. D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *Proc. ACM SIGCOMM*, Stanford, CA, Aug. 1988, pp. 106–14.
[14] R. L. Cruz, "A Calculus of Network Delay and a Note on Topologies of Interconnection Networks," Ph.D. thesis, Univ. of IL, July 1987.
[15] J. S. Turner, "New Directions in Communications (or Which Way to the Information Age?)," *IEEE Commun. Mag.*, vol. 24, no. 10, Oct. 1986, pp. 8–15.
[16] S. Keshav, "A Control-theoretic Approach to Flow Control," *Proc. ACM SIGCOMM*, Zurich, Switzerland, Sept. 1991, pp. 3–15.
[17] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *Proc. ACM SIGCOMM*, London, U.K., Aug. 1994. pp. 34–35.
[18] K. Ramakrishnan and S. Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IP," RFC 2481, IETF, Jan. 1999.
[19] K. K. Ramakrishnan and Raj Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *IEEE Trans. Comp. Sys.*, vol. 8, no. 2, May 1990, pp. 158–81.
[20] J. Postel, "Internet Control Message Protocol, " RFC (Standard) 792, IETF, Sept. 1981.
[21] W. Richard Stevens, *TCP/IP Illustrated Volume 1: The Protocols*, Reading, MA: Addison-Wesley, 1994.
[22] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Comp. Commun. Rev.*, vol. 24, no. 5, Oct. 1994, pp. 10–23.
[23] A. Charny, D. Clark, and R. Jain, "Congestion Control with Explicit Rate Indication," *Proc. ICC '95*, June 1995, pp. 1954–63.
[24] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*, Addison-Wesley.

[25] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case," *IEEE Net.*, vol. 1, no. 3, June 1993, pp. 344–57.
[26] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. ACM SIGCOMM*, Austin, TX, Sept. 1989, pp. 1–12.
[27] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case," *IEEE Net.*, vol. 2, no. 2, Apr. 1994, pp. 137–50.
[28] J. C. R. Bennett and H. Zhang, "WF2Q: Worst-case Fair Weighted Fair Queueing," *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 1996, pp. 120–28.
[29] S. J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," *Proc. IEEE INFOCOM*, Toronto, Canada, June 1994.
[30] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin," *ACM Comp. Commun. Rev.*, vol. 25, no. 4, Oct. 1995, pp. 231–42.
[31] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Trans. Net.*, vol. 3, no. 4, Aug. 1995, pp. 365–86.
[32] B. Suter *et al.*, "Efficient Active Queue Management for Internet Routers," *Proc. Eng. Conf. at Interop 98*, Las Vegas, NV, May 1998.
[33] S. Floyd and V. Jacobson, "Traffic Phase Effects in Packet-switched Gateways," *ACM Comp. Commun. Rev.*, vol. 21, no. 2, Apr. 1991, pp. 26–42.
[34] T. V. Lakshman, A. Neidhardt, and T. J. Ott, "The Drop from Front Strategy in TCP and in TCP over ATM," *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 1996.
[35] E. S. Hashem, "Analysis of Random Drop for Gateway Congestion Control," Tech. rep. MIT/LCS/TR-465, MIT Lab. for Comp. Sci., Cambridge, MA, Nov. 1989.
[36] B. Braden *et al.*, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, IETF, Apr. 1998.
[37] A. Mankin and K. Ramakrishnan, "Gateway Congestion Control Survey," RFC 1254, IETF, July 1991.
[38] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Net.*, vol. 1, no. 4, Aug. 1993, pp. 397–413.
[39] W. C. Feng et al., "A Self-Configuring RED Gateway," *Proc. IEEE INFOCOM*, NY, Mar. 1999.
[40] D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service," *IEEE/ACM Trans. Net.*, vol. 6, no. 4, Aug. 1998, pp. 362–73.
[41] IETF Integrated Services Working Group (intserv), http://www.ietf.org/html.charters/intserv-charter.html.
[42] IETF Differentiated Services Working Group (diffserv), http://www.ietf.org/html.charters/diffserv-charter.html.
[43] P. Ferguson and G. Huston, *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*, Wiley, 1998.
[44] R. Jain, "Myths about Congestion Control Management in High-speed Networks," *Internetworking: Res. and Experience*, vol. 3, June 1992, pp. 101–13.
[45] S. Shenker, "Fundamental Design Issues for the Future Internet," *IEEE JSAC*, vol. 13, no. 7, Sept. 1995, pp. 1176–88
[46] A. M. Odlyzko, "Paris Metro Pricing for the Internet," *Proc. ACM Conf. E-Commerce*, 1999, pp. 140–47.
[47] R. J. Gibbens and F. P. Kelly, "Resource Pricing and the Evolution of Congestion Control," *Automatica*, vol. 35, 1999.
[48] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," *ACM Comp. Commun. Rev.*, vol. 29, no. 4, Oct. 1999, pp. 109–20.

Biographies

PANOS GEVROS (p.gevros@cs.ucl.ac.uk) is a Ph.D. candidate in the Department of Computer Science University College London (UCL).He graduated from the University of Patras Department of Computer Engineering and Informatics in 1994, and received his M.Sc. degree in networks and data communications from the University of London in 1996. Since 1997 he has been a research fellow at the Department of Computer Science at UCL and has been working for several EU and DARPA funded research projects. His research interests are in models and mechanisms for resource management, congestion control, and differentiated services.

JON CROWCROFT [SM] (j.crowcroft@cs.ucl.ac.uk) is a professor of networked systems in the Department of Computer Science, UCL, where he is responsible for a number of European and U.S. funded research projects in multimedia communications. He has been working in these areas for 20 years. He graduated in physics from Trinity College, Cambridge University in 1979, and gained his M.Sc. in computing in 1981 and Ph.D. in 1993. He is a member of the ACM, and a Fellow of the British Computer Society, the IEE, and the Royal Academy of Engineering. He is a member of the IAB, and was general chair for ACM SIGCOMM 95–99. He is also on the editorial team for *ACM/IEEE Transactions on Networks and Computer Communications* as well as on the program committee for ACM SIGCOMM and IEEE INFOCOM. With Mark Handley, he is the co-author of *WWW:Beneath the Surf* (UCL Press); he also authored *Open Distributed Systems* (UCL Press/Artech House).A third book, *Internetworking Multimedia* (Taylor and Francis/Morgan Kaufman), was published in September 1999.

PETER T. KIRSTEIN (p.kirstein@cs.ucl.ac.uk) is professor of computer communications and director of research in the Department of Computer Science, UCL. He has been both technical and administrative director of many European and U.K. national projects. He has a B.A. from Cambridge University, M.Sc. and Ph.D. from Stanford University, and D.Sc. from the University of London. He was awarded the 1999 ACM SIGCOMM award for his contributions to the international development of the Internet. He has written some 160 papers and one book.

SALEEM N. BHATTI [M] (s.bhatti@cs.ucl.ac.uk) is a lecturer in the Department of Computer Science, UCL. His research interests are QoS (applications and networks), network management, network security, and mobile systems. He is a member of the ACM.