Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet

Vijay P. Kumar, T. V. Lakshman, and Dimitrios Stiliadis
Bell Laboratories, Lucent Technologies

ABSTRACT With the transformation of the Internet into a commercial infrastructure, the ability to provide differentiated services to users with widely varying requirements is rapidly becoming as important as meeting the massive increases in bandwidth demand. Hence, while deploying routers, switches, and transmission systems of ever increasing capacity, Internet service providers would also like to provide customer-specific differentiated services using the same shared network infrastructure. In this article, we describe router architectures that can support the two trends of rising bandwidth demand and rising demand for differentiated services. We focus on router mechanisms that can support differentiated services at a level not contemplated in proposals currently under consideration due to concern regarding their implementability at high speeds. We consider the types of differentiated services that service providers may want to offer and then discuss the mechanisms needed in routers to support them. We describe plausible implementations of these mechanisms (the scalability and performance of which have been demonstrated by implementation in a prototype system) and argue that it is technologically possible to considerably raise the level of differentiated services which service providers can offer their customers, and that it is not necessary to restrict differentiated services to rudimentary offerings even in very-high-speed networks.

The transformation of the Internet into an important and ubiquitous commercial infrastructure has not only created rapidly rising bandwidth demand but also significantly changed consumer expectations in terms of performance, security, and services. Consequently, service providers need to not only evolve their networks to higher and higher speeds, but also need to plan for the introduction of increasingly sophisticated services to address the varied requirements of different customers. At the same time, Internet service providers (ISPs) would like to maximize the sharing of the costly backbone infrastructure in a manner that enables them to control usage of network resources in accordance with service pricing and revenue potential. The two trends of rapidly rising bandwidth demand and rising need for differentiation has resulted in intense efforts to build fast packet forwarding engines and to define mechanisms for service differentiation.

The need to build fast forwarding engines is being addressed in a variety of ways. We discuss these in various sections of this article. The focus of the article, however, is on gigabit routers capable of not only fast forwarding but also providing customer-specific service differentiation. We consider a general and flexible model of service differentiation and do not restrict ourselves to any of the specific proposals for service differentiation being discussed in various industry and standards bodies. With a general model in mind, we discuss in detail the various router mechanisms necessary to provide differentiated services. In particular, we discuss architectural constraints imposed by the need to provide differentiated services instead of just fast forwarding, and also the three important components of packet filtering, buffer management, and scheduling necessary to support differentiated services at the network element level. For each of the mechanisms, we discuss implementation considerations drawing on experience gained from a prototype implementation. The theme is that at the network element level it is technologically viable to incorporate mechanisms which can provide customer-specific differentiated services even at very high speeds. Consequently, from the point of view of implementability there is no need to restrict service offerings to simple schemes encoded in the type of service (ToS) bits.

Differentiated Services

Consider what a company with multiple sites might expect, ideally, from an ISP providing connectivity among these sites. One possible scenario is that it would expect to receive, at the very least, a service similar to that obtained with a leased line network. This means that any two corporate sites communicating over the shared interconnecting backbone should perceive performance at least equal to that of a leased line of some minimum prespecified capacity. Traffic between sites should not be affected by the traffic of other users on the shared network. Queuing delays and congestion should occur only because of traffic generated by the individual company, not because of the overall load generated by others. Isolating traffic from different customers and providing minimum bandwidth guarantees in a customer-specific manner allow customers of ISP services to determine the bandwidth they require to satisfy their needs based on their own traffic requirements, just as they would with a leased line. They may want the additional flexibility of being able to specify the manner in which their internal traffic, from different sources, is allowed access to the available bandwidth. Furthermore, it may be desirable to define different levels of service for different types of traffic in a customer-dependent manner. For example, some customers may consider FTP or Web transfers to be low priority, so for them the ISP could aggregate multiple flows of these types. However, other customers may define voice over IP or database queries to be high priority, and therefore for them the ISP must ensure good performance by giving these traffic types priority or guaranteed minimum bandwidth from within that customer's available bandwidth.

0163-6804/98/$10.00 © 1998 IEEE

In another scenario, some customers may require extremely reliable and predictable performance for a small set of applications. They may indicate this requirement to the network as dynamic reservations of exact bandwidth along with specification of delay bounds. This indication may be by explicit signaling using a resource reservation protocol such as RSVP, or be done implicitly by some other means. The ISP's infrastructure must be capable of allowing end users to choose such a stringent, although possibly expensive, service if they require it. Considering scenarios such as the above, we can generate some guidelines regarding user and service provider requirements which can then be used in the design of router mechanisms for supporting differentiated services.

Customer Requirements

Customers would like the network to provide leased-line-like behavior while being able to use any additional available bandwidth. Since customers typically consolidate all existing network connections out of a site into a single connection to an ISP, they would like to have the added flexibility of dividing bandwidth available to them in a particular manner among their internal users and also by traffic type. This enables them to effectively support applications such as telephony, LAN interconnect, audio conferencing and video conferencing, Web hosting, Internet access, and so on, while at the same time being able to allocate available bandwidth flexibly among internal users.

Service Provider Requirements

While meeting the customer requirements, service providers want to maximize the sharing and multiplexing of their infrastructure to maximize their revenues. They must be able to take advantage of the possibility for statistical multiplexing.

Service providers should be able to identify traffic belonging to different customers so that customer-specific differentiation can be done. In conjunction with the need to provide differentiated services is the need to manage the associated revenue through customer-specific traffic accounting and billing. This is necessary so that different service-level assurances can be charged appropriately.

Furthermore, service providers should have the flexibility to distinguish themselves from other service providers by being able to tailor their service offerings in whatever competitive manner they choose.

All of these imply that routers must allow controlled resource sharing which permits service providers to maximize the utilization of their network while meeting diverse customer requirements.

The Current Internet Architecture

The prevalent Internet service model is the best-effort model (also known as the so-called *send and pray* model). This model does not permit users to obtain better service, no matter how critical their requirements are and no matter how much they may be willing to pay for better service. Clearly, with the increased use of the Internet for commercial purposes, this model of no differentiation is not satisfactory since the ISPs do not have the means to meet an existing market demand.

The Internet architecture has been successful up to this point, and the best-effort service model has been adequate. The current Internet architecture evolved from a network which was providing reliable connectivity, despite link failures, to thousands of users. Now, the Internet is an international network with millions of users. However, some of the control mechanisms that were originally implemented to prevent congestion collapse and depend on cooperative users are still in use. The network infrastructure did not incorporate any means for differentiation other than by severely underutilizing specific portions of the network. Overprovisioning network bandwidth and keeping the network lightly loaded in order to support differentiated services for select customers is not a cost-effective solution, and cannot be achieved at all times. Furthermore, for a commercialized network, relying on cooperation by many disparate users for congestion control is not desirable. It cannot be assumed that users will cooperatively slow down their transmission rates when significant congestion is detected. Assuming cooperative behavior leads to several problems, especially when the network has no means of isolating traffic from different users or enforcing cooperation. Among the problems are the following:

- End users ignore the expected and correct usage of TCP transmissions by disabling their flow-control mechanisms (either purposely or due to incorrect implementation). This degrades performance for complying users.
- Malicious users can easily degrade network performance by routing traffic through specific paths that increase local congestion. (An example is some of the recently reported denial-of-service attacks such as smurfing.)
- An increasing number of real-time applications use UDP without congestion control. For some applications, it is not optimal to adapt to congestion in a TCP-friendly manner. Applications may even increase their transmission rates during lossy periods by introducing more forward error correction. Such uncooperative behavior can quickly lead to poor performance for many users unless the network has sufficient mechanisms for isolating and penalizing such users.

Improving on Best Effort

It may be argued that best effort may be good enough with appropriate provisioning. Nevertheless, some form of differentiation is still desirable. When there is a focused overload to parts of the network (such as when a popular Web site is heavily accessed, or some event not accounted for in traffic engineering happens), routers must have mechanisms to treat different customers differently. When such events happen, there are just not enough resources available to give reasonable service to everybody.

Providing any form of differentiation usually requires the network to keep some state information. This requirement translates to increased memory requirements and processing power. The majority of the installed routers use architectures that will experience a dramatic decrease in performance if mechanisms are added to provide sophisticated differentiating features. It was safe to assume that with existing architectures, implementing any sophisticated functionality in the network was either impossible at the speeds required or would be prohibitively expensive. Therefore, the commonly held belief is that all of the sophistication should be in end systems and the core network as simple as possible.

However, there has been a shift toward incorporating more intelligence in the network. The current discussions about differentiated services within the IETF, for example, assume that the edge routers of a core network are smart enough to classify the packets of different users.

Requirements for Sophisticated Differentiated Services

Although traditional routers are limited in terms of their quality of service (QoS) and differentiation features, recent research and advances in hardware capabilities have provided

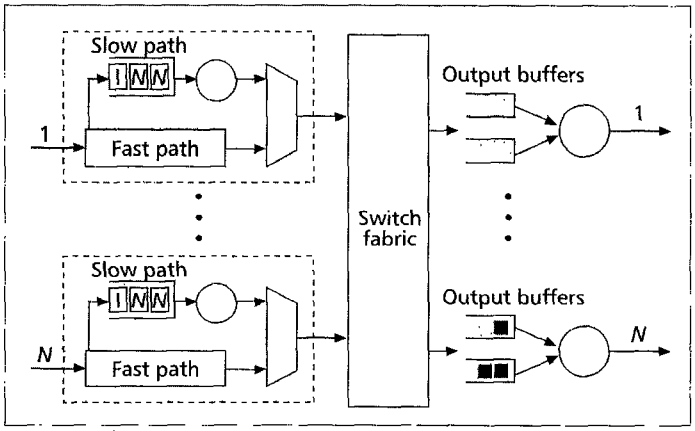


Figure 1. *Queuing model of a system that uses a cache-based architecture for packet processing and forwarding.*

mechanisms to overcome these limitations. From the router perspective, the tools for providing differentiated services are based on the following operations that can now be done at high speeds:

- Packet classification, which can distinguish packets and group them according to their different requirements
- Buffer management, which determines how much buffer space should be given to certain kinds of network traffic, and which packets should be discarded in case of congestion
- Packet scheduling, which decides the packet servicing order to meet the bandwidth and delay requirements of different types of traffic

High-speed implementation is made possible by the following:

- New developments in the areas of fair queuing and per-flow scheduling have resulted in simple and efficient algorithms for managing the complexity of scheduling decisions. These algorithms have been implemented in ATM switches, showing that they can be implemented at very high speeds even for packets as small as an ATM cell.
- State information can be managed using both aggregation and protocol-specific information. In the context of TCP, the argument has been that flows are too numerous in the backbone for per-flow information to be maintained. However, many of these TCP flows are not continually backlogged in the router queues. By providing support for tens of thousands of queues, which can be done without great expense, the performance requirements of a much higher number of TCP flows can be met by taking advantage of statistical multiplexing.
- The argument that route lookup and packet classification are expensive operations is becoming less valid as new research in these areas provides fast and efficient algorithms. New schemes can examine the complete information in the IP packet header and use it as a method of classification. When combined with fast switching fabrics, they can remove the processing bottlenecks of traditional routers.

Processing Models for Forwarding Engines

In this section, we first describe architectural features that are necessary in IP forwarding engines designed to provide differentiated services. Then, after showing why traditional router architectures cannot offer differentiated services, we present an example routing architecture that was designed specifically with the new requirements in mind.

Forwarding engines must, at the very least, be able to identify the context of packets, determine the next hop IP and link-layer addresses, and assign for each packet the appropriate buffering and output link bandwidth. The forwarding engine is the prime point of control in determining the manner in which packets are handled within the router.

The Requirement for Real-Time Operation

The first requirement for differentiated services is that the maximum delay for header processing must be no larger than the delay a packet from the service class with the least delay can experience. This is because it is the header processing which determines the service level to be accorded to a packet, and hence violation of service assurances prior to header processing cannot be allowed. Since the tolerable delay for some packets might be almost zero if we allow constant bit rate circuit-emulation-type flows, we get the following architectural constraint:

There should be no queuing before header processing.

This implies that packet header processing must be done at wire speeds and not be traffic-dependent. Traditionally, routers have used cache-based methods for forwarding packets. The justification is that packet arrivals are temporally correlated and can be grouped into connections or flows [1, 2] so that if a packet belonging to a new connection arrives, more packets belonging to that same connection can be expected to arrive in the near future. More precisely, the definition of a connection is:

A sequence of packets between the same source and destination with interarrival times of less than 60 s.

The reasoning in favor of cache-based architectures is that packets in the Internet are usually initiated by applications and form connections [3]. With this expected behavior, the first packet of an application can be processed through a slow path that analyzes the complete header. The header of the packet is then inserted into a cache or hash table together with the action that must be applied to this first packet as well as to all other packets of the same application. When subsequent packets of that flow arrive, the corresponding action can be determined directly from the cache or hash table.

There are three main problems associated with this architecture:

- In current backbone routers, the number of connections that are active at a given interface is extremely high. Recent studies have shown that an OC-3 interface might have an average of 256,000 connections active concurrently [4]. For this many connections the use of hardware caches is extremely difficult, especially if we consider the fact that a fully associative hardware cache is required. So caches of such size are most likely to be implemented as hash tables, since only hash tables can be scaled to these sizes. However, the $O(1)$ lookup time of a hash table is an average case result, and the worst-case performance of a hash table can be poor since multiple headers might hash into the same location. The number of bits in the packet headers that must be parsed is typically between 100 and 200 bits, and even hash tables are limited to only a couple of million entries. Thus, any hash function used must be able to randomly distribute 100–200-bit keys of the header to no more than 20–24 bits of hash index. Since there is no knowledge about the distribution of the header values of the packet that arrive to the router, the design of a good hash function is not trivial.
- Due to the large number of connections simultaneously active in a router and the fact that hash tables generally cannot guarantee good hashing under all arrival patterns, the performance of cache-based schemes is heavily traffic-dependent. If a large number of new connections arrive at the same time, the slow path of the

system that implements the complete header processing can be temporarily overloaded. This will result in queuing of packets before they are processed. When this happens, no intelligent mechanism can be applied for buffering and scheduling of these packets because without header processing there is no information available about either the destination of the packets or any other fields relevant to differentiation. So it is possible for congestion, packet dropping, and service quality violation to occur due to processing overload, not output link congestion. To better see this consider the model in Fig. 1. Packets arrive at the interfaces and are placed in a queue for processing. After the packet classification and next-hop lookup operations are performed, they are forwarded to the outgoing interfaces, where they are queued for transmission. With this architecture, some interfaces may be idle even when there are packets destined to those interfaces waiting in the input queues. For example, all packets destined for output 1 can be blocked in the slow path processing module behind packets that are destined to other outputs. Output 1 remains idle, although there are packets in the buffers available for transmission. Obviously, such a system will suffer from a type of head-of-line blocking that will limit the router throughput substantially. Note that the traditional head-of-line problem in input-buffered switches can be diminished if there is knowledge about the destination of more than one packet in the queue [36]. However, the fundamental problem of the system of Fig. 1 is that the destination or context of the packet is not actually known before the packet is processed. Thus, it is impossible to apply any intelligent queuing mechanisms in the input queues, and this form of head-of-line blocking cannot be eliminated.[1]

- A commercial Internet infrastructure should be robust and provide predictable performance at all times. Variations in the throughput of forwarding engines based on traffic patterns are undesirable. In addition, the network should not be vulnerable to attacks from malicious users. A malicious user or group of users discovering the limitations of the hash algorithms or caching techniques can generate traffic patterns that force the router to slow down and drop a large portion of the packets arriving at a particular interface.

Summarizing, we claim that packet queuing delays are acceptable only after header processing, if provisioning of differentiated services and robustness are important. This *no-queuing before processing* principle applies because it is the header processing operation that enables the router to determine the service level to be accorded to a particular packet. Hence, large queues formed while waiting for the packet processing operation can violate service levels for some connections, even before the router determines the service level to be accorded the connection. The implication this has on the design of forwarding engines is that it is the *worst-case performance* of the forwarding engine which determines the true maximum packet processing rate, not the average case performance (the averaging being done on traffic arrival patterns). If average case performance is used to determine supported packet processing speeds, buffering is needed before processing. Unless the variance in execution times of the header processing functions is very small, the delay in this preprocessing buffer can cause service assurances to be violated.

[1] *It is to be emphasized that the head-of-line blocking we refer to is not the traditional head-of-line blocking at the switch fabric. What we mean is that output interfaces may be idle even though there are packets destined to those interfaces waiting in the input header processing buffers.*

Criteria for Real-Time Packet Processing and System Constraints

We now list, based on the prior discussion, the criteria a differentiated-services-capable fast router must meet:

- The router must be fast enough to support gigabit links. ISPs envisage building networks with link capacities of 2.4 Gb/s and more. Any backbone router must be able to handle packets at these speeds.
- The forwarding engines of the routers must be able to process every packet arriving to the physical interfaces at wire speed. Recent traffic studies have shown that 75 percent of the packets are smaller than the typical TCP packet size of 552 bytes. In addition, nearly half the packets are 40–44 bytes in length, comprising TCP acknowledgments and control segments [4]. Since the forwarding engines cannot use buffering to absorb the variation in execution times, they must operate at wire speed when all packets are as small as 40 bytes. This means that the forwarding engines must have provably small worst-case execution times which are independent of traffic patterns.
- The processing engines of the backbone must be able to perform classification and packet filtering operations as well as next-hop lookup operations. In order to make the routers and networks manageable, both the packet filtering and route lookup operations must be based on aggregated rules such as classless interdomain routing (CIDR) aggregations [6, 7]. Memory accesses are expensive and are the dominant factor in determining the worst-case execution time.
- For operation at very high speeds, processing algorithms must be amenable to hardware implementation.

An Overview of Router Architectures

We give a brief overview of some of the most popular router architectures. Then we show how they fail to meet most of the requirements imposed by the dual needs of high capacity and differentiated services.

Single Processor–Shared Bus — The first generation of routers were based on a single general-purpose CPU and an intelligent real-time operating system. The choice of this architecture was based on the premise that protocols then were constantly changing, and multiprotocol operation meant that routers could not be optimized for a specific protocol. Establishing and maintaining connectivity was far more important than high forwarding capacity. These routers consisted of a general-purpose computer and multiple interface cards interconnected through a shared bus. Packets arriving at the interfaces were forwarded to the central processing engine which determined the next hop address and sent them back to the outgoing interfaces. Routing and other control protocols were also implemented in the same forwarding engine. Obviously, the performance of such a router depends heavily on the throughput of the shared bus and on the forwarding speed of the main processor. Since a single processor must perform multiple real-time operations, the selection of operating system is of critical importance, and a lot of emphasis was placed on the sophisticated design of the operating system. This architecture could not scale to meet the increasing throughput requirements of the interface cards.

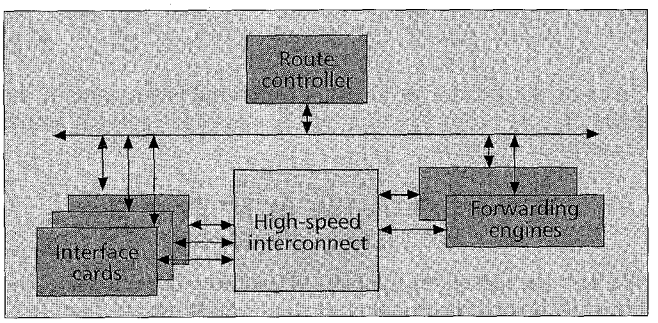


Figure 2. *Router architecture with multiple parallel forwarding engines that are separated from the interface cards. A load balancing mechanism attempts to equalize the processing load of all forwarding engines.*

Multiple Processors–Shared Bus — In first-generation routers, each packet was transmitted at least twice over the shared bus: once from the interface cards to the processor, and once from the processor to the outgoing interface card. The straightforward improvement introduced in the second generation of routers was to distribute the forwarding computation. Interface cards became more intelligent with the addition of fast processors and caches. This allowed them to process packets locally some of the time. The first packet of a connection is forwarded to the main forwarding engine. A cache entry is added to the interface card cache, and this allows subsequent packets of the same connection to be forwarded directly between the interfaces. The cache entry is on a per-connection or per-route basis [8]. The argument for the latter was that even if the number of connections is very large, the number of routes needed is probably not large. However, in the network core where the highest forwarding speeds are required, it is unlikely that this will be true since packets are likely to be destined to many parts of the network and not be localized to specific subnets. Indeed, it is more likely that packets are distributed to every other router in the network. This architecture clearly has traffic-dependent throughput, and the shared bus is still a bottleneck.

As memory sizes increased and cost dropped, the distributed interface cards were enhanced with larger memories, and complete forwarding tables were copied to each interface card. This approach further enhanced the performance of these routers. However, the shared bus architecture and general-purpose CPUs cannot scale to higher-capacity links and with traffic-pattern-independent throughput.

Multiple Processors–Space Switching — The third generation of routers were designed to alleviate the two obvious bottlenecks of the second generation of routers. The congested shared bus was replaced by a crossbar switch providing sufficient bandwidth for transmitting packets between interface cards and allowing throughput to be increased by several orders of magnitude. With the forwarding path between interface cards not being the bottleneck anymore, the new bottleneck is packet processing. Most third-generation router architectures used general-purpose CPU designs that could not handle large link capacities.

Shared Parallel Processors–Space Switching — A nice concept for parallelizing packet processing to be able to scale processing speeds considerably was introduced in [9, 10]. The basic observation is that it is highly unlikely that all interfaces will be bottlenecked at the same time. Hence, sharing of the forwarding engines can increase the port density of the router. This shared processor architecture is presented in Fig. 2. The forwarding engines are only responsible for resolving next-hop addresses. The results of this next-hop address resolution are sent back to the interface cards which subsequently forward the packets to the outgoing interfaces. Note that only packet headers are forwarded to the forwarding engines. This eliminates an unnecessary packet payload transfer over the switch fabric. Packet payloads are always directly transferred between the interface cards, and they never go to either the forwarding engines or the controller unless specifically destined to them.

It is possible that if enough forwarding engines are added to such a system, it can reach the necessary forwarding capacities for high-speed backbones. However, the assumption on which this architecture is based is that a single processor is unlikely to be able to handle the peak throughput of a particular interface, and therefore a processing pool shared by multiple interfaces is necessary. The time required to process each packet depends on the actual load of each forwarding engine. Any load balancing mechanism must be done on a per-connection basis; out-of-order transmissions can trigger TCP fast retransmits and degrade performance. Therefore, the timescale on which load balancing is done (the timescale of connection interarrival times) can be an order of magnitude or more different from the timescale of packet arrivals. If, after a connection has been assigned to a processor, the load of the processor increases (because it is hard to predict the traffic generated by each connection when load balancing decisions are made), packets could be queued for long periods before processing and cause violation of service assurances for that connection. It can be claimed that if the number of forwarding engines is sufficiently large, there is a high probability that one forwarding engine will always be free. Although this is true, it is not clear how cost-effective it is. Note that a copy of the forwarding databases must be kept at each forwarding engine and must all be kept consistent. Among the architectures discussed so far, this is the most promising.[2]

Next, we present a router architecture that can scale to very large forwarding speeds while providing worst-case traffic-independent bounds on processing times.

Optimizing Packet Processing

It is evident that increasing link capacities and the need for differentiated services stretch processor-based architectures to the limit. The multiprocessor architecture described in the previous section attempts to address these issues by using several forwarding engines, each of which can perform packet processing for packets arriving on particular sets of links. In this section, we develop a more economical and efficient solution based on a functional partition of packet processing with each processing element optimized to a specific task and operating at wire speed in a traffic-independent manner.

The first step necessary for this approach is to identify the main functions that make up the forwarding process. An example enumeration of functions is the following (each of these are discussed in detail in subsequent sections):

- Buffer and forward packets through some switching fabric.
- Apply filtering and packet classification.
- Determine the next hop of the packet. Use the forwarding databases created by the routing protocols to determine to which outgoing interface the packet is to be sent and what the specific link-layer addresses are.

[2] *This architecture has also been studied for use in processing signaling protocols. Useful load balancing schemes and a detailed performance analysis are presented in [11].*

- Queue the packet in an appropriate queue based on both the classification decisions and the route table lookup.
- Schedule packet transmission on outgoing links to meet QoS requirements.

The common aspect of all these processing functions is that they use as inputs the packet header and some data structures created by protocols such as routing protocols. When the same processing element executes all of the above functions, it must have access to both the packet and to all these data structures. Thus, when multiple processors are used for forwarding packets and every packet is completely processed by one of them, these complete data structures must be up to date in every processor. Obviously, the synchronization problems for achieving this are nontrivial (see [11] for a performance analysis of this problem in the context of processing signaling protocols). In addition, since each packet requires accesses to different databases, there is no locality of reference in the data structures required by the processors. Since all processors have to keep all the forwarding information and data structures, the memories required are large and thus generally slow.

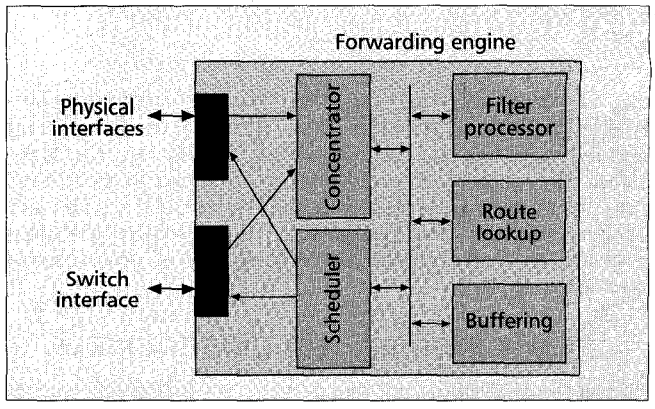
There is an easy way, however, to exploit the special requirements of packet processing to increase performance and reduce cost. The architecture for this is shown in Fig. 3. Packets arrive at the interfaces, and their processing is functionally split among multiple processors so that each processor handles one of the packet processing functions necessary for forwarding. For the forwarding functions described above, five processors are required. Since each processor performs only one type of function, each processor need only access the data structures associated with a particular function. It is not necessary to keep all information related to forwarding in every processor. Thus, the memory required for each processor is much smaller and hence can be much faster and cheaper. Specializing the task each processor performs makes it possible to get a high locality of reference for program memory accesses.

Additional advantages of such an architecture are:

- There is no connection-level assignment of processing units to connections. For the previously discussed multiprocessor architecture, such an assignment is needed for load balancing, and this has several disadvantages that were pointed out. With this architecture, flow identification is only required for storing packets in the output link queues.
- The importance of the operating system is diminished. Each processor is a single processing element that is dedicated to one function.
- When higher throughput is desirable, application-specific integrated circuits (ASICs) can replace processing elements easily. The one-function-only nature of each processor enables efficient hardware implementation.

Note here that a combination of the above architecture with the multiple shared processors architecture is possible as well, and may be very cost-efficient. One minor problem might be that communication between interfaces and processors might add some additional constant latency to the packet forwarding. As we will see in subsequent sections, each of these individual functions can be implemented at wire speeds in a traffic-independent manner even at backbone link rates. Hence, this architecture supports the twin requirements of both high speed and differentiated services.

A variant of the above architecture was developed and implemented as part of the Bell Laboratories Router project. A multigigabit router was designed where each interface card can process up to 1 million packets/s. The processing elements were implemented using field-programmable gate arrays (FPGAs), and the system speed was only 33 MHz. Obtaining such high throughput with differentiated services features using a low clock speed is evidence of the scalability of this architecture.



■ **Figure 3.** *Processing architecture where parallelism is achieved by partitioning forwarding functions among different processors.*

ROUTE TABLE LOOKUP

For a long time, route table lookup was considered one of the most challenging operations performed during the forwarding process. The problem is defined as searching through a database of destination prefixes and locating the longest prefix that matches the destination address of a given packet. Longest prefix matching was introduced as a consequence of the requirement to increase the number of networks addressed through CIDR [6].

The first approaches to longest prefix matching used radix trees or modified Patricia trees [12, 13] combined with hash tables. These lookup algorithms have complexity proportional to the number of address bits, which for IPv4 is only 32. Hence, even with these algorithms a route lookup can complete in less than 1 μs using 30 ns memories, thus resulting in a forwarding rate of 1 million packets/s. When the radix trees are changed to use a radix greater than 2, the time required is even lower, although at the expense of some wasted memory space.

Early implementations of routers, however, could not afford such expensive computations. Thus, most routers relied on either connection or route caches [8, 14]. Cache-based architectures, however, result in unpredictable and traffic-dependent performance. Furthermore, with the rapid Internet growth it is not clear anymore how effective caches are in network backbones.

Recent research has resulted in even faster solutions. The stratified tree algorithms proposed by Van Emde Boas in [15, 16] were utilized by De Berg, Van Kreveld, and Snoeyink to propose algorithms for one-dimensional point location when the problem is restricted to discrete valued ranges with $O(\log N)$ complexity, where N is the number of bits required to represent the range endpoints [17]. The same technique was recently used in the context of route table lookups in [18]. The main idea is to first create a perfect hash table of prefixes for each prefix length. A binary search among all prefix lengths, using the hash tables for searches among prefixes of a particular length, can find the longest prefix match in $O(\log N)$ time. Although the algorithm has very fast execution times, calculating perfect hashes can be slow and can slow down updates.

Another elegant approach was proposed in [19]. The basic principle is to create a small, compressed data structure, exploiting the sparseness of actual entries in the space of all possible routing table entries, which represents

large lookup tables using a small amount of memory. This results in a lower number of memory accesses to a fast memory and hence faster lookups. In addition, the algorithm does not have to calculate expensive perfect hash functions, although updates to the route table are still not easily done. Another approach for implementing the compression and minimizing the complexity of updates was presented in [20].

PACKET FILTERING AND CLASSIFICATION

One of the most important requirements for forwarding engines that support differentiated services is the ability to identify the context of packets and apply the actions necessary to satisfy service requirements. Mechanisms to support the above functions are termed *packet filtering* or *packet classification*.

The traditional application of packet filters has been to provide firewall and security functions. However, another important application, in support of differentiated services is the identification and classification of packets originated by specific sites or customers to provide the resources necessary for meeting customer-specific differentiated services requirements. Large-scale packet filtering functionality enables both edge routers, as in [21], and core routers to support differentiated services that are more flexible and customer-specific than those restricted to service offerings based on interpretation of the ToS bits alone [22].

Packet filters should parse a large portion of the packet header, including information concerning transport protocols, before forwarding decisions are made.[3] The parsing is based on a set of rules defined by either network management software or real-time reservation protocols such as RSVP. The actions packet filters may apply can include both the traditional security functions, such as dropping of unauthorized packets, redirection of packets to proxy servers, and such, as well as actions related to queuing and scheduling, and routing decisions based on a criterion other than destination address.

Two desirable attributes for packet filtering are:

- Filter rules must apply to ranges of addresses, port numbers, or protocols, and should not be restricted to exact matches. This allows rules to apply to aggregates and keeps the number of rules to be specified manageable. If filter algorithms can only handle exact matches, preprocessing must translate ranges in filter rules to exact values. This is unfeasible since the size of ranges grows exponentially with the length of the packet field on which the ranges are defined.
- Rules must be assigned explicit priorities in order to resolve conflicts if rules overlap.

THE GENERAL PACKET CLASSIFICATION PROBLEM

The general packet classification problem can be viewed as a point location problem in multidimensional space, where given a set of n d-dimensional objects (which represent the filter rules) and a point in a d-dimensional space (which represents the arriving packet), the problem is to find the object that contains the point. This is a classic problem in computational geometry, and numerous results have been reported in the literature [23–26]. When considering the general case of $d > 3$ dimensions, as in the problem of packet classification, the best algorithms considering time or space have either an $O(\log^{d-1} n)$ complexity with $O(n)$ space, or an $O(\log n)$ time complexity with $O(n^d)$ space [40]. Although algorithms with these complexity bounds are useful in many applications, they are not directly useful for packet filtering. To illustrate this, let us assume that we would like the router to be able to process 1000 rules of 5 dimensions within 1 μs (to sustain 1 million packets/s throughput). An algorithm with $\log^4 n$ execution time and $O(n)$ space requires 10,000 memory accesses/packet. This is impractical with any current technology. If we use an $O(\log n)$ time and $O(n^4)$ space algorithm, the space requirement becomes prohibitively large since it is in the range of 1000 Gbytes. Furthermore, none of the above algorithms addresses the problem of arbitrary overlaps. However, the above results do not mean that packet filtering cannot be done at high speeds given the constraints of the problem when applied to routers. Below, we sketch ideas that were used for a prototype implementation. Interested readers are referred to [27] for details of the implemented algorithm.

A simple approach to the problem of multidimensional search, as used for packet filtering, is to use decomposable search. Here the idea is to state the original query as the intersection of a few numbers of simpler queries. The challenge then, for instance to obtain a poly-logarithmic solution, is to decompose the problem such that the intersection step does not take more time than the required bound. However, as was pointed out before, even a $\log^4 n$ solution for five-dimensional packet filtering is not practical for our application where n can be in the thousands. Therefore, we need to employ parallelism of some sort. Moreover, we require simple elemental operations to make the algorithm amenable to hardware implementation. Instead of algorithms with the best asymptotic bounds, we are interested in decomposing the queries such that subquery intersection can be done fast (as per our memory-access cost metric) for n in the thousands and memory word lengths that are feasible with current technology.

We will illustrate the algorithm in [27] with an example in two dimensions. Rules are represented by two-dimensional rectangles that can arbitrarily overlap. The preprocessing step of the algorithm projects the edges of the rectangles to the corresponding axis. In the worst case, the projection results in a maximum of $2n + 1$ intervals on each dimension. We next associate a set of rules with each dimension. A rule belongs in the set if and only if the corresponding rectangle overlaps with the interval to which the set corresponds. Assume that a packet arrives at the system. During the first online step, we locate the intervals in both axes that contain this point. In the second step, we use the sets to locate the highest-priority rectangle that covers this point by a simple intersection operation.

The algorithm has been implemented in five dimensions in a high-speed router prototype using a single FPGA device and synchronous SRAM chips supporting up to 512 rules and processing 1 million packets/s in the worst case. This is achievable despite the device being run at a very low speed of 33 MHz. Since we used the same memory chips as those used in the L2 caches of PCs, the cost of the memories is low. The device can be used as a coprocessor next to a general-purpose processor that handles all the other parts of IP forwarding or firewall functions. An improved algorithm presented in [27] can be used to process 8000 filter rules at 1 million packets/s using a 66 Mhz clock.

CLASSIFICATION IN TWO DIMENSIONS

The 2D classification problem is of increasing importance in the evolving Internet architecture. Although RSVP, or similar reservation protocols, can offer end-to-end QoS guarantees for specific flows, it is not clear whether such a reservation-

[3] *In the industry forwarding decisions based on transport protocol information are referred to as layer-4 forwarding.*

based model can be scaled for operation over high-speed backbones where a very large number of flows can be concurrently active. An alternative approach that has been proposed is to route aggregated flows along specific traffic-engineered paths. This directed routing is based not only on the destination address of packets, but on the source address as well [28]. RSVP or similar reservation protocols can be used for setting the aggregation and routing rules [28, 29]. The key mechanism needed to support such a scheme in a high-speed core router is a two-dimensional classification or lookup scheme that determines the next hop, and the associated resource allocations, for each packet as a function of both the source and destination addresses. Two-dimensional classification is useful not only for setting up traffic-engineered source-destination paths, but also for multicast forwarding, which requires lookups based on both the source address and multicast groups [30, 31].

The two-dimensional lookup problem is a restricted case of the general classification problem where the rules in at least one of the two dimensions (source or destination address) are defined in terms of CIDR prefixes. Rules can still have arbritrary overlaps, and priority among rules is used to resolve conflicts. For IP routers, we are interested in solutions that can scale to hundreds of thousands of entries. Also, as for all other operations, we are interested in only worst-case performance of the algorithms since we want to avoid queuing for header processing in order to provide QoS guarantees.

A fast algorithm for this problem is presented in [27]. It is shown that for a number of possible prefix lengths of 32 and $n = 2^{18} = 256{,}000$ filter rules, the number of memory accesses is about 50, which makes it possible to process at 1 million packets/s with a less than 100 MHz clock speed. Combining fast filtering in many dimensions with source–destination-based routing widens the range of options feasible for evolving the current best-effort Internet to the Internet of the future, capable of providing customized differentiated services. Specifically, there may be no need to restrict filtering to the edges or to very simple operations such as using only the ToS bits in the IP packet header.

Resource Management

The previous sections discussed how processing power, one of the scarce router resources, should be managed to meet the new differentiated services requirements of the Internet. An important principle was "*no queuing before processing*" and consequent architectural choices for worst-case engineering. The other critical resource is output link bandwidth. Coupled with management of output link bandwidth is buffer allocation. This section is focused on bandwidth and buffer management. Buffer and bandwidth allocation must be jointly addressed. A scheduler that does dynamic bandwidth allocation merely gives opportunities to connections for link access. If a connection does not have any packets waiting to be transmitted, this opportunity is wasted (although some schedulers retain these lost opportunities as credits for future use). For adaptive connections, it is especially important that provision of transmission opportunities (i.e., scheduling) be coupled with proper buffer allocation which ensures that these connections use the link access opportunities and so achieve a desired level of link sharing. A desired level of link sharing is set for classes of packets using administrative means or by dynamic reservations such as RSVP signaling. Packet filtering mechanisms map each arriving packet into one of these classes. Schedulers determine the sequence of packet transmissions on the link taking into account the desired link sharing, the packets from different classes currently in the system, and a history of recent service sequence (which in a fair queuing scheduler is summarized in a so-called system potential, to be elaborated on later). We distinguish between two types of connections that impact scheduling.

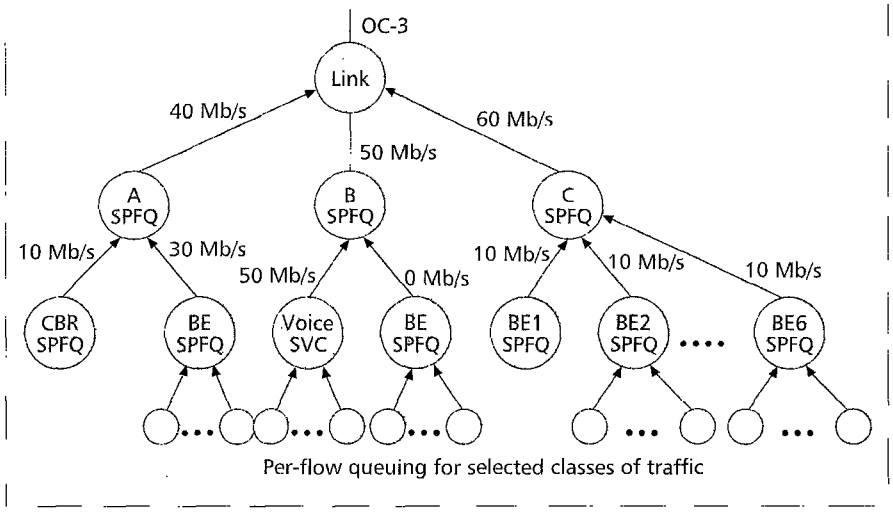
Nonadaptive — Connections whose packet arrival rates are independent of past scheduling decisions. Examples are open-loop UDP traffic such as video or audio streams. The bandwidth requirements for these connections are set by provisioning mechanisms, or established by signaling. Provisioning is usually done at an aggregate level; that is, the requirement might be that a given amount of bandwidth must be ensured for connections of a certain type between two sites in a network. Bandwidth establishment by signaling can be done using RSVP which allows setting up aggregate and flow-level end-to-end reservations. Irrespective of the mechanism, resource control mechanisms are required in every node to enforce the reservations.

Adaptive Flows — These adapt their transmission rates based on scheduling decisions made in the recent past, such as those made a round-trip time ago. An example is TCP connections which constantly probe for available bandwidth by increasing their transmission rates in response to available network bandwidth. Adaptive flows, in particular TCP flows which are loss-sensitive in their adaptation, require careful buffer management in conjunction with scheduling. The need for TCP-aware buffer management has been pointed out in [34–37]. We discuss TCP-aware buffer management in detail later in the article.

A further requirement imposed by the commercialization of the Internet is that user-specific differentiated service models must be supported since the same infrastructure is to be used to serve different user requirements. The state-of-the-art candidate scheme for bandwidth allocation to meet differentiated services requirements is by hierarchical link sharing using hierarchical weighted fair queuing. This is discussed next.

Hierarchical Link Sharing

Hierarchical link sharing is a flexible resource management scheme that allows partitioning of resources at several levels of aggregation. It was initially proposed in [38, 39] and later expanded on in [40]. Hierarchical link sharing partitions bandwidth into a hierarchy of classes, where each class is defined to be a set of flows with some specific properties, such as packets that belong to the same protocol, or packets traveling between specific subnets. The levels of the hierarchy can be made to include, at the lowest level, packets from an individual application. The hierarchical link sharing model, as proposed in [39], is illustrated in Fig. 4. Here, the main idea is that link capacity is partitioned among different administrative domains, and within each domain it can be further partitioned based on application types. Going a step further, the bandwidth of each application type can be further partitioned among individual flows of that type. A different view of this model is that the link capacity is partitioned between classes of traffic, and the bandwidth of each traffic class is further partitioned between administrative domains or individual flows. Our prototype implements Hierarchical Weighted Fair Queuing (HWFQ), which has the extra advantage of using the same schemes across all hierarchy levels and thus making the implementation simpler. The model presented in [40] was extended, using the theory of shaped rate-proportional servers [41], so that each node in the hierarchy can be any shaped rate-proportional



■ **Figure 4.** *Application of hierarchical link sharing. Each node can be either a shaped virtual clock (SVC) or a start-potential-based worst-case fair queueing (SPFQ) scheduler.*

server. Then a node in the hierarchy may share excess bandwidth with the other nodes at the same level, or it may shape traffic to a particular bandwidth allocation. In our implementation, each node in the hierarchy is allowed to use a shaped virtual clock, which provides ideal shaping of the traffic, or use Starting Potential-Based Fair Queuing, which allows fair distribution of the bandwidth among competing classes. Both algorithms are shaped rate-proportional servers and use the exact same data structures [41].

The advantage of this general mechanism is best explained with an example. Assume that for one administrative domain we want to support two types of traffic: constant bit rate (CBR) and best-effort. For a second administrative domain, we want to give voice traffic full priority over best-effort traffic. Finally, for a third administrative domain, we would like different levels of best-effort service. The assignment of schedulers and weights is shown in Fig. 5.

We assign a virtual clock scheduler to the CBR traffic, and a fair queuing scheduler to the best-effort traffic of the first domain (A). This mechanism allows the best-effort traffic to get any of the excess bandwidth available from other classes. For domain B, we assign all the bandwidth to the high-priority voice traffic. As a result, best-effort traffic will only be serviced when voice traffic is not present. For the third domain, we assign different weights to the different levels of best-effort service.

The advantage of the above scheme is that we can use the same data structures and algorithms to provide a variety of services to different administrative domains or applications.

The next question is whether we should do any aggregation at all at the flow level. If it is possible to implement per-flow queuing for a large number of flows, we can derive many advantages. Clearly, flow isolation is a benefit of per-flow queuing. Enhanced TCP performance by combining per-flow queuing and TCP-aware buffer management is another benefit. This is discussed in detail later.

Scalability of Per-Flow Queuing

The prevailing assumption is that per-flow queuing is prohibitively expensive because routers must keep state for all flows, and there can be hundreds of thousands of active flows in network backbones. The notion of active flow used is that a flow is active as long as the interarrival time of any two packets between the same source and destination is less than 60 s [42]. This notion is useful when the purpose is to study mechanisms which ensure that only a small number of packets are forwarded to the slow path and that most of them are handled through a cached fast path. The problem with using this as the basis for maintaining flow state for scheduling purposes in per-flow queuing schemes is that this is overly conservative. Maintaining state for each such active flow is not necessary to provide the guarantees of fair queuing except in some extreme situations.

There are mainly two types of state a per-flow queuing mechanism must maintain:

- The weights of individual flows
- Information about the service offered to flows

When providing differentiated services, in most cases weights are established for a set of flows with given characteristics, not for individual flows. An example might be differentiating between TCP ftp-like flows and TCP telnet-like flows. The packet filtering function performs the differentiation and will typically use filter rules that classify many flows into a class rather than have a separate class per flow. Once the weight for a class is established, each flow must use a different queue to enable the per-flow queuing mechanism to isolate flows and guarantee the desired differentiation. As discussed below, the state information necessary to provide this isolation is less than that required to maintain state for every active flow.

Ideal Weighted Fair Queuing — Weighted Fair Queuing schedulers [43, 44], emulate a fluid-flow system within the constraints of a packet system. In an ideal fluid flow system packets are infinitely divisible, and all backlogged flows are simultaneously served with a rate proportional to the assigned weights. In such a scheduler, the instantaneous service offered to an individual flow is only determined by the weight assigned to this flow and the weights assigned to all other flows that are currently backlogged. Thus, there is no need for the scheduler to keep state for any flow that is not currently backlogged. Since the number of backlogged flows is limited by the available buffering, the number of flows for which state has to be maintained is also limited to numbers determined by buffer availability.

Non-Ideal Weighted Fair Queuing — For packet fair queuing schedulers to emulate an ideal fluid flow system, they have to maintain all the state information of the fluid flow system as well as information related to the state of the packet system. This is because a packet scheduler cannot emulate a fluid flow system exactly since packets from different flows cannot be served at the same time. In the worst case, the set of flows backlogged in the reference fluid system and the set of flows actually backlogged in the packet system may be completely disjoint [45]. In this case, the maximum state information a packet fair queuing system must maintain is approximately twice that of the fluid flow system. This can still be much lower than the number of active flows.

In [45], a class of schedulers has been defined that use simple mechanisms to extract the state of the fluid flow system from the actual state of the packet system. This information is

used to schedule packets in the packet system [32, 45]. These schedulers associate a function with each flow or connection, called the *connection potential*, that tracks the normalized service offered to the connection while the connection is active.[4] A system potential [32, 45] is used to keep track of the normalized service offered to all flows during the current system busy period. When a flow is newly backlogged, its flow potential is calculated as the maximum of its previous value and the value of the system potential. For some types of schedulers, there is a bound on the difference between the connection potential of any nonbacklogged flow and the system potential. In these schedulers, in addition to the backlogged flows, state must be maintained for a small subset of flows that were backlogged during the immediate past. Furthermore, in the case of Self-Clocked Fair Queuing (SCFQ), the system potential is always larger than the connection potential of any nonbacklogged flow [32]. Thus, for SCFQ, state must be maintained only for the backlogged flows (see [36] for a more detailed discussion).

Summarizing, for an Ideal Weighted Fair Queuing scheduler or SCFQ scheduler, in the worst case, the number of flows is equal to the number of packets in the system. As an example, assume that a router supports an OC-12 link (622 Mb/s bandwidth). Assume that the average packet size is 256 bytes. Then, assuming a buffer size equivalent to 200ms, it is easy to verify that the maximum number of flows which can be backlogged in the router is no more than 64,000, which is a very reasonable number within current technology limitations.

TCP-Aware Buffer Management — Since TCP-controlled traffic constitutes a significant component of Internet traffic, it is natural to incorporate mechanisms in routers that enhance TCP performance. The following general criteria act as a guideline. For long TCP flows (transfers much larger than the obtained bandwidth-delay product) we would like to keep the link share as close to the desired one as possible.[5] For short transfers, we would like small queuing delays for some TCP flows such as telnet flows. We would like to achieve good TCP performance even in the presence of cross-traffic which is not TCP-controlled, uncontrolled, or controlled in a manner that is unfair to TCP sources. Also, we would like to keep the link utilization as high as possible.

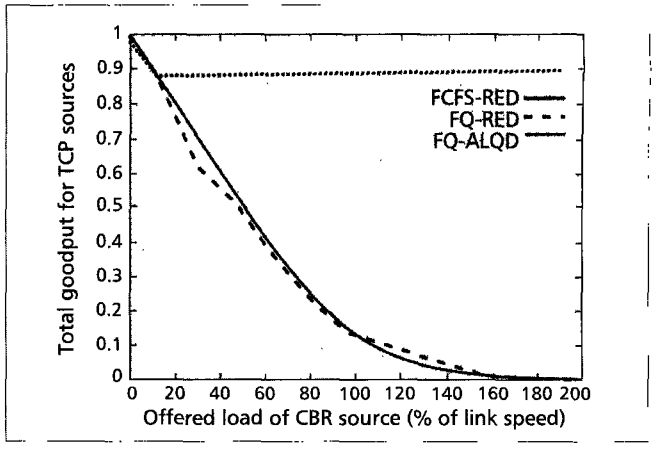
TCP Observations

Traffic Burstiness — TCP creates bursty network traffic. This is because data transmission in TCP is ack clocked, and acks bunch together in FIFO queues to create burstiness [46, 47]. Also, TCP slow-start is very bursty due to the doubling of the window sizes every round-trip time. Losses in the slow-start phase can lead to very poor throughput, and buffering at least equal to approximately one-third the bandwidth delay product is needed to avoid losses in the slow-start phase [48]. However, to keep throughput high, especially for long transfers, a generally accepted rule of thumb is to have buffering equal to at least one bandwidth delay product.

With such large buffers, packets from quasi-delay-sensitive applications, like telnet, may queue up behind a large slow-start burst from a file transfer and experience large queuing delays despite the use of mechanisms like RED [49]. Hence, it would be good to separate telnet-like sources from others.

[4] *Normalized service is defined as the ratio of offered service to allocated bandwidth.*

[5] *For ease of exposition, we only explain the case where link sharing within a class is equal (i.e., fair link sharing).*



■ **Figure 5.** *Aggregate throughput for 10 TCP sources (2 to 160 ms RTT) sharing a link with a loss-insensitive CBR source.*

Unfairness — It is known that TCP is inherently unfair to connections with long round-trip times [50], and the unfairness can sometimes be as bad as the inverse square of round-trip times [48]. This implies the use of active TCP-aware buffer management in routers to alleviate unfairness as suggested in [34].

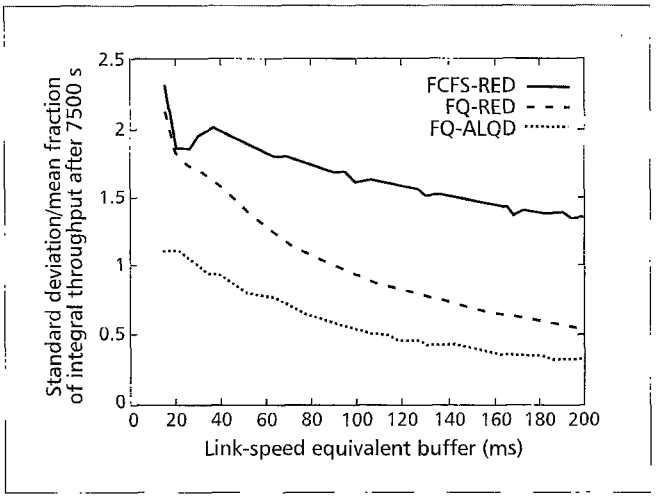
Synchronization — Another reason for the use of TCP-aware buffer management is that with drop-tail queuing, TCP windows can synchronize, leading to poor and oscillatory link utilization [51]. It is necessary to reduce synchronization by appropriate buffer management.

Random Loss Sensitivity — Since TCP assumes that every loss is a congestion loss and reduces its transmission rate drastically, TCP throughput is very vulnerable to loss. For a fixed loss rate, the throughput goes down as the inverse square of the bandwidth-delay product [48, 52]. This loss sensitivity is further increased if there is a slow reverse path or the ack path is congested. The sensitivity increase is proportional to the ratio of the transmission time of ack packets in the reverse path to that of data packets in the forward path [53]. Random losses can be caused by transient fast (faster than round-trip times) fluctuations in open-loop traffic such as uncontrolled UDP traffic. Buffer management must protect TCP sources from losses caused by TCP-unfriendly sources (i.e., sources that do not react to losses by reducing transmissions as fast as TCP).

Is Per-Flow Fair Queuing Sufficient for Achieving TCP Performance Goals? — Per-flow fair queuing provides fair opportunities-to-transmit. However, if a flow does not have a backlog (due to small windows, for instance) often enough then fair opportunities-to-transmit do not translate to fair link sharing [35]. Proper buffer management is needed in conjunction with fair queuing to ensure that TCP flows share the available bandwidth in a fair manner (or in a controlled unfair manner if desired).

The link-sharing goals for TCP are to achieve:

- High throughput
- Fairness in bandwidth sharing among competing TCP connections
- Protection of conforming connections from malicious users
- Reduce the well-known ack-compression phenomenon for TCP [46] which causes network traffic to be bursty even for smooth sources
- Provision of low latency to telnet-like applications which are delay-sensitive and use TCP



■ **Figure 6.** *Coefficient of variation versus buffer size for many TCP connections with differing round-trip times.*

Is Buffer Management Alone Sufficient? — In the absence of per-flow queuing, control of link sharing is achieved by buffer management and packet discard schemes like Random Early Detection (RED) and drop-from-front. Although some of the above goals can be achieved, the lack of per-flow state hampers the degree to which the goals are achieved. However, the above goals can be achieved much better with a combination of per-flow queuing and appropriate buffer management [35].

TCP-Aware Buffer Management for Use with Fair Queuing — Merely dividing available buffer space and using RED on each queue is not appealing since buffer space is wasted drastically. For good buffer usage, we would like to share buffer space between different flows. Using RED on this shared buffer space breaks the natural flow isolation that per-flow queuing provides (see [35] for an example) and allows TCP connections to be affected by TCP-unfriendly sources. Also, fair queuing with uncontrolled buffer usage by each flow leads to unfairness.

A solution is to give each flow a nominal reserved buffer space. These nominal allocations can be set in proportion to weights (or ratio of bandwidth to round-trip times if they are known). Each flow's buffer usage can exceed reservation if unused buffer space is available. However, if the total buffer occupancy exceeds a global threshold and a new packet arrives, some packet already in the buffer must be pushed out. The candidate queues eligible for pushout are those queues whose occupancy is above their nominal allocation. From this candidate set, we pick the queue that has the most excess occupancy (i.e., longest deviation from nominal location). The reason we pick this longest queue is that TCP flows with short round-trip times and high bandwidth usage are likely to have the longest queues above nominal since their windows grow faster than flows with longer round-trip times. We alleviate unfairness to long round-trip time connections by dropping from this longest queue. Also, TCP-unfriendly flows are likely to have long queues. Within the longest queue, we use drop-from-front [54].

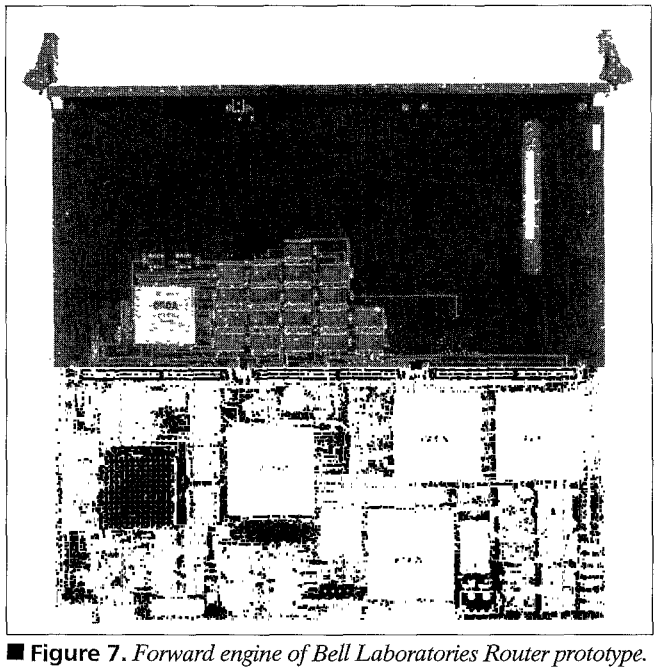
Drop-from-front triggers TCP's fast retransmit feature more quickly, reducing the length of congestion episodes and improving link utilization. It improves fairness even with FIFO queuing, and reduces chances of windows synchronizing. It is also a useful mechanism for dropping acks [53] (in case of ack congestion) since TCP acks are cumulative. Also, when used with fair queuing it almost eliminates the chance of retransmitted packets being lost [35]. Loss of retransmitted packets leads to expensive TCP timeouts and consequent loss of throughput.

This buffer management scheme protects TCP sources from TCP-unfriendly flows. Hence, with this buffer management there is no need for other applications (such as reliable multicast or RTP applications) to have TCP-friendly congestion control. Those applications can adapt their traffic in whatever manner is preferable to them without having to comply with TCP's congestion control mechanism. TCP-aware buffer management eliminates the need for other applications to be TCP-aware. Another useful aspect is that this buffer management allows TCP to have larger initial windows (hence speeding up completion of short transfers) without affecting other TCP flows.

Protection from TCP-Unfriendly Sources — The protection that the above buffer management scheme gives to TCP sources is illustrated in Fig. 5.

Figure 5 shows the fraction of the link bandwidth used by 10 persistent TCP sources sharing a link with a loss-insensitive TCP-unfriendly source [35]. The sending rate of this unfriendly source is varied from zero to twice the link bandwidth. With a consistent overload produced by the TCP-unfriendly source, the buffer management and dropping strategy have a stronger impact on bandwidth sharing than the scheduling policy itself. When the TCP-unfriendly source increases its rate, for both FCFS and fair queuing with global random early detection (RED) there is very similar TCP-source performance degradation. However, with longest queue drop policies (marked ALQD in the figure to denote an approximated longest queue implementation), an FQ scheduler is able to almost perfectly maintain the guaranteed rate of 1/11 per flow, irrespective of whether the flow is TCP-like or totally loss-insensitive.

Figure 6 shows the coefficient of variation of the throughput of different TCP flows when they share a common link with fair queuing and with RED or the buffer management scheme proposed in the previous section. Clearly, the proposed buffer management scheme has better fairness than RED.



■ **Figure 7.** *Forward engine of Bell Laboratories Router prototype.*

Implementation Issues

It has been a long-standing assumption that implementation of large-scale per-flow queuing and hierarchical link sharing is not possible at reasonable cost with current technologies. However, prototype implementations of gigabit routers with per-flow queuing and hierarchical link sharing has shown that it is indeed possible to implement these mechanisms at the necessary scale at high speeds. A primary limitation in applying intelligent buffer management and scheduling mechanisms was that processing elements were performing all functions for header processing, which limited throughput as functionality increased. Distribution of processing on a functional basis and not on a per-packet basis, as shown in Fig. 3, allows throughput to be increased considerably while increasing functionality as well. Furthermore, the state information needed for per-flow queuing is not prohibitive, as has been assumed.

In the Bell Laboratories Router prototype, shown in Fig. 7, we have demonstrated that per-flow queuing and hierarchical link sharing can be implemented, on a scale necessary for backbone routers, using a single FPGA device running at 33 MHz and supporting output link capacities up to 1 Mb/s.

Switch Fabric

Switch fabric design is a very well studied area, especially in the context of ATM networks [5, 55–59], and a detailed discussion of switch fabric design issues is omitted for brevity. The key point to keep in mind is that transport through the fabric must be done in a service-preserving manner.

Discussion and Conclusions

With the Internet's ongoing evolution from a best-effort network to a network with service differentiation, an important topic of interest is the type of differentiated services that service providers may want to provide. One possibility is to encode in the ToS bits a set of canonic differentiated service types and incorporate only these mechanisms into the network infrastructure. However, it is not clear whether this simple and evolutionary approach can satisfy differentiated services demands.

The point of view taken in this article is that it is possible to incorporate in routers using current technology, mechanisms which, in addition to enabling simple differentiated services, as proposed recently, also enables differentiated services that are more sophisticated in their ability to meet specific customer needs. The required mechanisms are:

- All header processing done at wire speed: "no queuing before processing"
- Packet classification with range matching on many fields for a large number of rules, extending route lookups to source-and-destination-based lookups
- Flexible and general scheduling and buffer management techniques
- QoS-capable switch fabric
- A mechanism for flow isolation

Using these mechanisms, routers can aggregate customers in a very flexible manner, isolate individual customers' traffic, and allocate resources in a customizable manner. This permits each ISP to customize its service offerings to suit its customer demands by offering services such as VPNs with customized service differentiaton, virtual leased lines, and so on. Restricting router functionality to simple differentiated services possibilities due to technology constraints is not necessary.

Acknowledgments

The authors would like to thank members of the Bell Laboratories Router team wo were responsible for building a prototype router encompassing many of the ideas discussed in this article. We would like to thank, in particular, R. Arunachalam, S. Rathnavelu, K. J. Singh, B. Suter, and H. Tzeng for their many contributions.

References

[1] K. C. Claffy, "Internet Traffic Characterization," Ph.D. thesis, UC San Diego, 1994.
[2] K. Claffy, C. Polyzos, and H. W. Braun, "Application of Sampling Methodologies to Network Traffic Characterization," *Proc. ACM SIGCOMM '93*, Sept. 1993, pp. 194–203.
[3] R. Jain and S. Routhier, "Packet Trains — Measurements and a New Model for Computer Network Traffic," *IEEE JSAC*, vol. 4, 1986, pp. 986–95.
[4] K. Thomson, G. J. Miller, and R. Wilder, "Wide-Area Traffic Patterns and Characteristics," *IEEE Network*, Dec. 1997.
[5] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *Proc. INFOCOM '96*, Mar. 1996, pp. 296–302.
[6] V. Fuller *et al.*, "Classless Inter-Domain Routing," RFC1519, ftp://ds.internic.net/rfc/rfc1519.txt, June 1993.
[7] L. Zhang *et al.*, "RSVP: A New Resource Reservation Protocol," *IEEE Network*, vol. 7, no. 5, Sept. 1993, pp. 8–18.
[8] C. Partridge, "Locality and Route Caches," NSF Wksp. on Internet Statistics Measurement and Analysis, San Diego, CA, Feb. 1996.
[9] A. Asthana *et al.*, "Design of a Gigabit IP Router," Tech. rep. 11251-911105-09TM, AT&T Bell Labs., Nov. 1991.
[10] A. Asthana *et al.*, "Towards a Gigabit IP Router," *J. High-Speed Networks*, vol. 1, no. 4, 1992.
[11] D. Ghosal, T. V. Lakshman, and Y. Huang, "Parallel Architectures for Processing High-Speed Network Signaling Protocols," *IEEE/ACM Trans. Networking*, Dec. 1995, pp. 716–28.
[12] K. Sklower, "A Tree-Based Routing Table for Berkeley Unix," Tech. rep., UC Berkeley, 1993.
[13] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on Longest-Matching Prefixes, *IEEE/ACM Trans. Networking*, vol. 4, no. 1, Feb. 1996, pp. 86–97.
[14] D. C. Feldmeier, "Improving gateway performance with a routing-table cache," *Proc. of IEEE INFOCOM '88*, New Orleans, LI, March 1988.
[15] P. Van Emde Boas, "Preserving Order in a Forest in Less than Logarithmic Time," *Proc. 16th IEEE Conf. Foundations of Comp. Sci.*, 1975, pp. 75–84.
[16] P. Van Emde Boas, R. Kaas, and E. Zijlstra, "Design and Implementation of an Efficient Priority Queue," *Math. Sys. Theory*, vol. 10, 1977, pp. 99–127.
[17] M. De Berg, M. van Kreveld, and J. Snoeyink, "Two- and Three-Dimensional Point Location in Rectangular Subdivisions," *J. Algorithms*, vol. 18, 1995, pp. 256–77.
[18] M. Waldvogel *et al.*, "Scalable High Speed IP Routing Lookup," *Proc. ACM SIGCOMM '97*, France, Sept. 1997.
[19] M. Degermark *et al.*, "Small Forwarding Tables for Fast Routing Lookups," *Proc. ACM SIGCOMM '97*, France, Sept. 1997.
[20] H.-Y. Tzeng, "Longest Prefix Search Using Compressed Trees," Tech. rep. TM, Bell Labs., under submission, 1997.
[21] D. Clark, "Service Allocation Profiles," Internet draft, http://www.internic.net/internet-drafts/draft-clark-diff-svc-alloc-00.txt, 1997.
[22] K. Nichols and S. Blake, "Differentiated Services Operational Model and Definitions," Internet draft, Feb. 1998. http://ds.internic.net/internet-drafts/draft-nichols-dsopdef-00.txt.
[23] B. Chazelle, "How to Search in History," *Info. and Control*, vol. 64, 1985, pp. 77–99.
[24] B. Chazelle and J. Friedman, "Point Location among Hyperplanes and Unidirectional Ray Shooting," *Comp. Geometry: Theory and Apps.*, vol. 4, 1994, pp. 53–62.
[25] K. L. Clarkson, "New Applications of Random Sampling in Computational Geometry," *Discrete & Comp. Geometry*, vol. 2, 1987, pp. 195–222.
[26] M. H. Overmars and A. F. van der Stappen, "Range Searching and Point Location among Fat Objects," *J. Algorithms*, vol. 21, no. 3, 1996, pp. 629–56.
[27] T. V. Lakshman and D. Stiliadis, "Packet Classification Algorithms for Gigabit Internet Routers," Tech. Rep. 113470-980202-02T, Lucent Technologies-Bell Labs., Jan. 1998; also to appear, *Proc. ACM SIGCOMM '98*.
[28] T. Li and Y. Rekhter, "Provider Architecture for Differentiated Services and Traffic Engineering (PASTE)," Internet draft, http://www.internic.net/internet-drafts/draft-li-paste-00.txt, 1998.
[29] J. Boyle, RSVP Extensions for CIDR Aggregated Data Flows, Internet draft, http://www.internic.net/internet-drafts/draft-ietf-rsvp-cidr-ext-01.txt, 1997.
[30] D. Estrin *et al.*, "Protocol Independent Multicast — Sparse Mode," Protocol spec. RFC 2117, June 1997.

[31] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol," RFC 1075, ftp://ds.internic.net/rfc/rfc1075.txt, June 1993.
[32] S. J. Golestani, "A Self-Clocked Fair Queuing Scheme for Broadband Applications," *Proc. IEEE INFOCOM '94*, Apr. 1994, pp. 636–46.
[33] D. Stiliadis and A. Varma, "Design and Analysis of Frame-Based Fair Queuing: A New Traffic Scheduling Algorithm for Packet-Switched Networks," *Proc. ACM SIGMETRICS '96*, http://www.cse.ucsc.edu/research/hsnlab/publications/, May 1996, pp. 104–15.
[34] B. Braded *et al.*, "Recommendations on Queue Management and Congestion Avoidance in the Internet," Internet draft, Mar. 1997.
[35] B. Suter *et al.*, "Design Considerations for Supporting TCP with Per-Flow Queuing," *Proc. IEEE INFOCOM '98*, San Francisco, CA, 1998.
[36] B. Suter *et al.*, "Efficient Active Queue Management for Internet routers," *Proc. Interop Engineers Conf.*, Las Vegas, NV, May 1998.
[37] D. Lin and R. Morris, "Dynamics of Random Early Detection," *Proc. ACM SIGCOMM '97*, Sept. 1997.
[38] D. D. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proc. ACM SIGCOMM '92*, Aug. 1992, pp. 14–26.
[39] S. Floyd and V. Jacobson," Link Sharing and Resource Management Models for Packet Networks," *IEEE/ACM Trans. Networking*, vol. 3, no. 4, Aug. 1995, pp. 365–86.
[40] J. C. R. Bennett and H. Zhang, "Hierarchical Packet Fair Queuing Algorithms," *Proc. ACM SIGCOMM '96*, Palo Alto, CA, Aug. 1996, pp. 143–56.
[41] D. Stiliadis and A. Varma, "A General Methodology for Designing Efficient Traffic Scheduling and Shaping Algorithms," *Proc. IEEE INFOCOM '97*, 1997.
[42] P. Newman, T. Tylon, and G. Minshall, "Flow Labelled IP: A Connectionless Approach to ATM," *Proc. IEEE INFOCOM '96*, San Francisco, CA, Apr. 1996, pp. 1251–60.
[43] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control — The Single Node Case," *Proc. INFOCOM '92*, vol. 2, May 1992, pp. 915–24.
[44] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," *Internetworking: Res. and Experience*, vol. 1, no. 1, 1990, pp. 3–26.
[45] D. Stiliadis and A. Varma, "Rate-Proportional Servers: A Design Methodology for Fair Queuing Algorithms," *IEEE/ACM Trans. Networking*, Apr. 1998.
[46] S. Shenker, L. Zhang, and D. D. Clark, "Some Observations on the Dynamics of a Congestion Control Algorithm," *Comp. Commun. Rev.*, Oct. 1990, pp. 30–39.
[47] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," *Proc. ACM SIGCOMM '91*, 1991, pp. 133–47.
[48] T. V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Trans. Networking*, June 1997.
[49] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Networking*, Aug. 1993.
[50] S. Floyd and V. Jacobson," "On Traffic Phase Effects in Packet Switched Gateways," *Internetworking: Res. and Experience*, vol. 3, no. 3, Sept. 1992, pp. 115–56.
[51] L. Zhang, "A New Architecture for Packet Switching Network Protocols," Ph.D. thesis, MIT Comp. Sci., Cambridge, MA, 1989.
[52] T. J. Ott, M. Mathis, and J. H. B. Kemperman, "The Stationary Behavior of Ideal TCP Congestion Avoidance," ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps, 1996.
[53] T. V. Lakshman, U. Madhow, and B. Suter, "Window-Based Error Recovery and Flow Control with a Slow Acknowledgment Channel: A Study of TCP/IP Performance," *Proc. IEEE INFOCOM '97*, Kobe, Japan, Apr. 1997.
[54] T. V. Lakshman, A. Neidhart, and T. J. Ott, "The Drop-from-Front Strategy in TCP over ATM and Its Interworking with Other Control Features," *Proc. IEEE INFOCOM '96*, San Francisco, CA, 1996, pp. 1242–50.
[55] F. A. Tobagi, "Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks," *Proc. IEEE*, vol. 78, no. 1, Nov. 1990, pp. 133–67.
[56] F. M. Chiussi, J. G. Kneuer, and V. P. Kumar, "Low-Cost Scalable Switching Solutions for Broadband Networks," *IEEE Commun. Mag.*, vol. 35, no. 12, Dec. 1997, pp. 36–43.
[57] F. M. Chiussi, "Design, Performance and Implementation of a Three-Stage Banyan-Based Architecture with Input and Output Buffers for Large Fast Packet Switches," Tech. rep. CSL-TR-93-573, CSL, Stanford, CA, June 1993.
[58] T. E. Anderson *et al.*, "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. Comp. Sys.*, vol. 11, no. 4, Nov. 1993, pp. 319–52; also, Digital Systems Research Center tech. rep. no. 99.
[59] D. Stiliadis and A. Varma, "Providing Bandwidth Guarantees in an Input-Buffered Crossbar Switch," *Proc. INFOCOM '95*, Apr. 1995.

BIOGRAPHIES

VIJAY P. KUMAR (Vijay@bell-labs.com) is head of High Speed Networks Research at Bell Laboratories, Lucent Technologies. He is responsible for research and development in algorithms, architectures, protocols, chips, and systems for high-speed data networking. He is also responsible for making next-generation routing engines into products. He obtained his Bachelor of Engineering degree in electronics and communication engineering from Osmania University, Hyderabad, India, in 1980, and M.S. and Ph.D. degrees in electrical and computer engineering from the University of Iowa, Iowa City, in 1982 and 1985, respectively. He has been with Bell Laboratories since 1985, where he has conducted and led research activities in fast packet switching, VLSI communication architectures, fault-tolerant networking, and VLSI yield enhancement. His work has resulted in several ATM chipsets and a gigabit routing engine for differentiated services.

T. V. LAKSHMAN (lakshman@research.bell-labs.com) received his Ph.D. degree in computer science from the University of Maryland, College Park, in 1986. Prior to that he received a Master's degree from the Department of Physics, Indian Institute of Technology, Bangalore, India. From 1986 to 1995 he was at Bellcore, where he was most recently a senior research scientist and technical project manager in the Information Networking Research Laboratory. He is currently with the High Speed Networks Research Department at Bell Laboratories. His recent research has been in issues related to traffic characterization and provision of quality of service for video services, architectures and algorithms for gigabit IP routers, end-to-end flow control in high-speed networks, traffic shaping and policing, switch scheduling, and parallel architectures for fast signaling and connection management in high-speed networks. His current research interests are in the areas of high-speed networking, distributed computing, and multimedia systems. He is a corecipient of the 1995 ACM Sigmetrics/Performance Conference Outstanding Paper Award. He is an editor of the *IEEE/ACM Transactions on Networking*.

DIMITRIOS STILIADIS (stiliadi@bell-labs.com) received his Diploma in computer engineering and informatics from the University of Patras, Greece, in 1992. He received M.S. and Ph.D. degrees in computer engineering from the University of California at Santa Cruz in 1994 and 1996, respectively. He is currently a member of technical staff in High-Speed Networks Research of Bell Laboratories, where he has been leading the architecture and design of a gigabit router supporting differentiated services. His current research interests include traffic scheduling, network performance analysis, and architectures of fast forwarding engines.