Issues and Trends in Router Design

S. Keshav and Rosen Sharma
Cornell University

ABSTRACT Future routers must not only forward packets at high speeds, but also deal with nontrivial issues such as scheduling support for differential services, heterogeneous link technologies, and backward compatibility with a wide range of packet formats and routing protocols. In this article, the authors outline the design issues facing the next generation of backbone, enterprise, and access routers. The authors also present a survey of recent advances in router design, identifying important trends, concluding with a selection of open issues.

Routers knit together the constituent networks of the global Internet, creating the illusion of a unified whole. While their primary role is to transfer packets from a set of input links to a set of output links, they must also deal with heterogeneous link technologies, provide scheduling support for differential service, and participate in complex distributed algorithms to generate globally coherent routing tables. These demands, along with an insatiable need for bandwidth in the Internet, complicate their design.

Routers are found at every level in the Internet. Routers in *access* networks allow homes and small businesses to connect to an Internet service provider (ISP). Routers in *enterprise* networks link tens of thousands of computers within a campus or an enterprise. Routers in the *backbone* are not usually directly accessible to end systems. Instead, they link together ISPs and enterprise networks with long distance trunks. The rapid growth of the Internet has created different challenges for routers in backbone, enterprise, and access networks. The backbone needs routers capable of routing at high speeds on a few links. Enterprise routers should have low cost per port and a large number of ports, be easy to configure, and support quality of service (QoS). Finally, access routers should support many heterogeneous high-speed ports and a variety of protocols at each port, and try to bypass the central office voice switch.

This article presents the design issues and trends that arise in these three classes of routers. The following section describes the structure of a generic router. The section after that discusses design issues in backbone, enterprise, and access routers. We then present some recent advances and trends in router design. Finally, we conclude with a description of some open problems. We note that our main topic of discussion is packet forwarding; routing protocols, which create the forwarding tables, are dealt with only in passing.

Components of a Router

Figure 1 abstracts the architecture of a generic router. A generic router has four components: input ports, output ports, a switching fabric, and a routing processor. An *input port* is the point of attachment for a physical link and is the point of entry for incoming packets. Ports are instantiated on *line cards*, which typically support 4, 8, or 16 ports. The *switching fabric* interconnects input ports with output ports. We classify a router as input-queued or output-queued depending on the relative speed of the input ports and the switching fabric. If the switching fabric has a bandwidth greater than the sum of the bandwidths of the input ports, packets are queued only at the outputs, and the router is called an output-queued router. Otherwise, queues may build up at the inputs, and the router is called an input-queued router. An *output port* stores packets and schedules them for service on an output link. Finally, the *routing processor* participates in routing protocols and creates a *forwarding table* that is used in packet forwarding. We now discuss the components of this generic router in more detail.

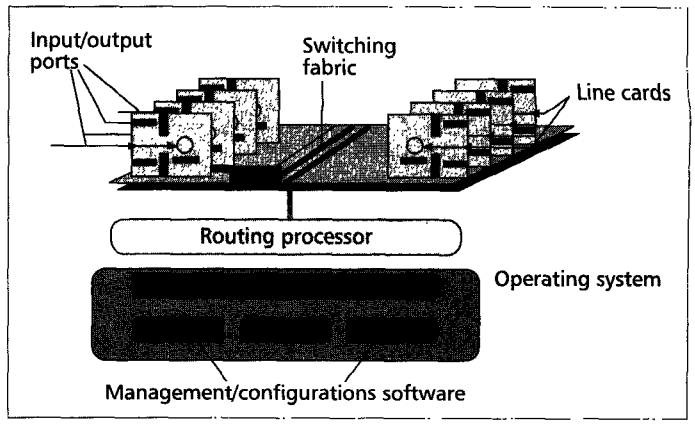
An input port provides several functions. First, it carries out data link layer encapsulation and decapsulation. Second, it may also have the intelligence to look up an incoming packet's destination address in its forwarding table to determine its destination port (this is called *route lookup*). The route lookup algorithm can be implemented using custom hardware, or each line card may be equipped with a general-purpose processor. Third, in order to provide QoS guarantees, a port may need to classify packets into predefined service classes. Fourth, a port may need to run data-link-level protocols such as Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP), or network-level protocols such as Point-to-Point Transmission Protocol (PPTP). Once the route lookup is done, the packet needs to be sent to the output port using the switching fabric. If the router is input-queued, several input ports must share the fabric: the final function of an input port is to participate in arbitration protocols to share this common resource.

The switching fabric can be implemented using many different techniques (for a detailed survey see [1]). The most common switch fabric technologies in use today are buses, crossbars, and shared memories. The simplest switch fabric is a bus that links all the input and output ports. However, a bus is limited in capacity by its capacitance and the arbitration overhead for sharing this single critical resource. Unlike a bus, a crossbar provides multiple simultaneous data paths through the fabric. A crossbar can be thought of as $2N$ buses linked by $N * N$ crosspoints: if a crosspoint is on, data on an input bus is made available to an output bus; otherwise, it is not. However, a scheduler must turn on and off crosspoints for each set of packets transferred in parallel across the crossbar. Thus, the scheduler limits the speed of a crossbar fabric. In a shared-memory router, incoming packets are stored in a shared memory and only pointers to packets are switched. This increases switching capacity. However, the speed of the switch is limited to the speed at which we can access memory. Unfortunately, unlike memory size, which doubles every 18 months, memory access times decline only around 5 percent each year. This is an intrinsic limitation of shared-memory switch fabrics.

0163-6804/98/$10.00 © 1998 IEEE

Output ports store packets before they are transmitted on the output link. They can implement sophisticated scheduling algorithms to support priorities and guarantees. Like input ports, output ports also need to support data link layer encapsulation and decapsulation, and a variety of higher-level protocols. More details on packet scheduling can be found in [2].

The routing processor computes the forwarding table, implements routing protocols, and runs the software to configure and manage the router. It also handles any packet whose destination address cannot be found in the forwarding table in the line card.



■ **Figure 1.** *Architecture of a router.*

Design Issues

With this description of a generic router in hand, in this section we turn our attention to design issues for backbone, enterprise, and access routers.

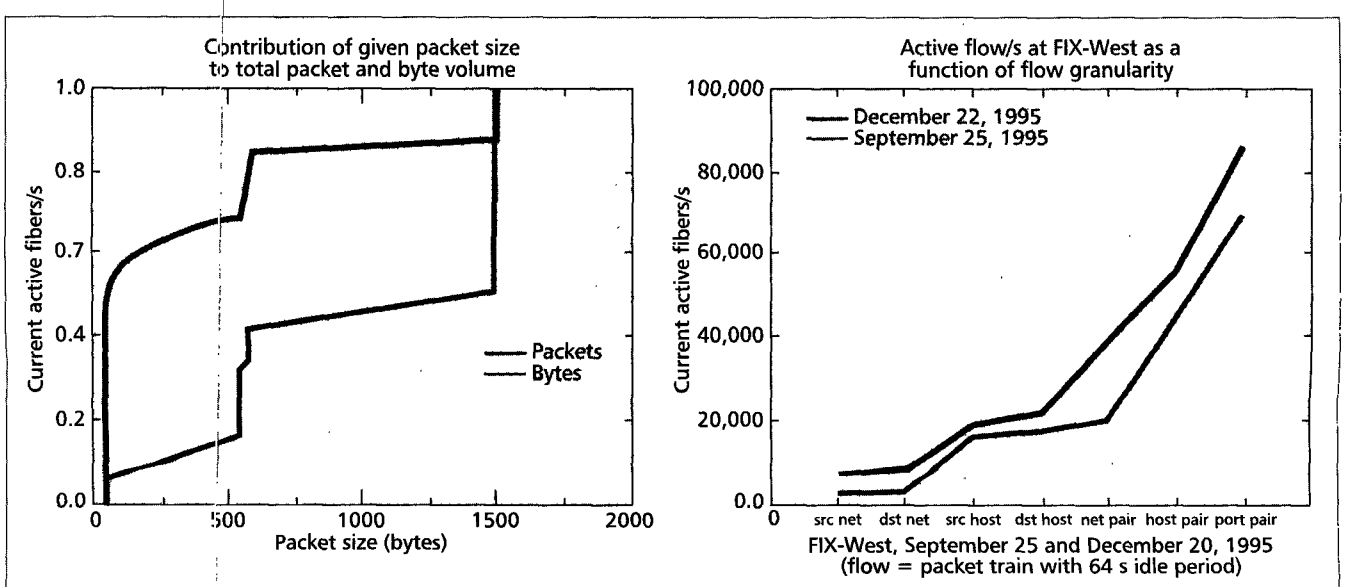
Backbone Routers

The Internet currently has a few tens of backbones that each serve up to a few thousand smaller networks. Backbone routers interconnect enterprise networks, so their cost is shared among a large customer base. Moreover, the cost of wide-area transmission links is currently so high that cost is a secondary issue in the design of backbone routers. The primary issues are reliability and speed.

Hardware reliability in backbone routers can be achieved using much the same techniques as in telephone switches: hot spares, dual power supplies, and duplicate data paths through the routers. These are standard in all high-end backbone routers, and we will not consider these issues in any detail. Instead, we focus on techniques to achieve high-speed routing.

The major performance bottleneck in backbone IP routers is the time taken to look up a route in the forwarding table. On receiving a packet, an input port looks up the packet's destination address in its forwarding table to determine the packet's destination port. The forwarding table stores routing entries of the form <*network address/mask, port*>. On receiving a packet with address A, the port conceptually cycles through all its forwarding entries. For each entry, the router masks A with the mask stored with that entry, and if this matches the corresponding network address, adds *port* to the set of candidate destination ports. The selected destination is the candidate port corresponding to the longest mask (we call this the entry with the *longest prefix match*). For example, consider a router that has the following entries in its routing table: {<128.32.1.5/16, 1>, <128.32.225.0/18, 3>, <128.0.0.0/8, 5>}. A packet with a destination address 128.32.195.1 matches all three entries, so the set of candidate destination ports for this packet is {1, 3, 5}. However, port 3 corresponds to the routing entry with the longest mask (i.e., 18). Therefore, the destination port of the packet is port 3. This example outlines the two reasons why looking up a route is hard. First, the routing table may contain many thousands of entries. Thus, it is highly inefficient to match an incoming packet serially with every entry. Indeed, to achieve high routing speeds, we should tightly bound the worst-case time required to look up a destination port. Second, an incoming packet may match multiple routing entries. We must find, among matching entries, the one with the longest match.

The cost of route lookups increases if packets are small or packets are routed to a large number of destinations, so a cache of frequently visited destinations becomes ineffective. We now present some representative measurements to evaluate the situation in current backbones. Figure 2a shows the



■ **Figure 2.** *a) Packet size distribution in a backbone router (data courtesy MCI, 25 June 1997, OC-3 backbone link); b) number of active flows in a backbone router.*

packet size distribution from a 5 min trace collected on a backbone router in the MCI backbone [3]. We see that approximately 40 percent packets are 40 bytes long (these correspond to TCP acknowledgment packets). This distribution implies that current backbone routers must perform a large number of route lookups every second. One metric of route diversity is the size of the forwarding cache if routes are kept in the cache for a given time period. In a trace obtained in 1995, Fig. 2b [4], we find that if routes are kept in a cache for only 64 s since their last use, we still need a cache of 64,000 entries.

Input-queued and output-queued routers share the route lookup bottleneck, but each has an additional performance bottleneck that the other does not. Recall that output-queued switches must run the switch fabric at a speed greater than the sum of the speeds of the incoming links. While this can be solved by building third-generation interconnection networks, that still leaves the problem of storing packets rapidly in output buffers. The rate at which the output buffer can be accessed is limited by DRAM or SRAM access times. These ultimately limit the speed at which an output-queued router can be run. One way to get around this problem is to place all queuing at the input. However, with this approach an arbiter must resolve contention for the switching fabric and output queue. It is hard to design arbiters that run at high speeds and can also fairly schedule the switch fabric and output line [5, 6]. We discuss strategies for speeding up output-queued routers later.

In addition to overcoming bottlenecks in performance at individual routers, there is an additional design issue that is often ignored. We believe that the stability and reliability of routing protocol implementations critically affect the scalability of the Internet. Little is known about the stability of networks where routers run different versions of the same protocol, or worse, run different protocols altogether. Thus, even slight changes in network configuration can lead to serious and nearly undetectable problems. For example, the descriptions of filters to export (import) routes from an interior gateway protocol to BGP is the cause of many a routing problems on the Internet. Recent studies have shown that the routing oscillations which characterize the current Internet are often the result of small bugs in protocol implementation or router misconfiguration [7].

Enterprise Routers

Enterprise or campus networks interconnect end systems. Unlike backbone networks, where speed comes first and cost is a secondary issue, their primary goal is to provide connectivity to a large number of endpoints as cheaply as possible. Moreover, support for different service qualities is desirable because this would allow QoS guarantees at least for traffic confined to the local area. Most enterprise networks are currently built from Ethernet segments connected by hubs or bridges. These devices are cheap and easy to install, and require no configuration. However, not only does the performance of a network built with hubs and bridges degrade with the network's size, but there is usually little support for service differentiation. In contrast, a network built with routers partitions the machines into multiple collision domains and therefore scales better with the size of the network. Besides most routers support some form of service differentiation, at least allowing multiple priority levels. Routers, however, tend to be more expensive per port and require extensive configuration before they can be used. The challenge, therefore, is to build enterprise routers that have a low cost per port and a large number of ports, are easy to configure, and support QoS.

Enterprise routers have several additional design requirements. Unlike backbone networks, enterprise networks may carry a significant amount of multicast and broadcast traffic. Thus, they must carry multicast traffic efficiently. Backbone routers tend to support only the IP protocol. In contrast, enterprise networks, which must deal with legacy LAN technologies, must support multiple protocols, including IP, IPX, and Vines. They must also support features such as firewalls, traffic filters, extensive administrative and security policies, and virtual LANs. Finally, unlike backbones, which link a handful of trunks, enterprise routers must provide a large number of ports. Thus, enterprise router designers must solve the conflicting design goals of providing a rich feature set at each port, and reducing the cost per port. This is not an easy task!

Access Routers

Access networks link customers at home or in a small business with an ISP. Access networks have traditionally been little more than modem pools attached to terminal concentrators, serving a large number of slow-speed dialup connections. However, this simple model of the access network is changing. First, access networks are beginning to use a variety of access technologies such as high-speed modems, ADSL, and cable modems. Second, the use of telephone lines for accessing the Internet from home has increased the load on the phone network. The long connection holding time for Internet dialup connections creates problems for phone switches. Thus, there is considerable pressure on access networks to be aware of the underlying telephone network, and to try to bypass the voice switch for data calls. Third, access routers are beginning to provide not just a SLIP or PPP connection, but also virtual private network protocols such as PPTP and IPSec. These protocols need to be run at every router port. Finally, technologies such as asymmetric digital subscriber loop (ADSL) will soon

IP vs. ATM

ATM was designed from the ground up to enable cheap switches. Small virtual connection identifiers (VCIs) are rapidly looked up, and fixed-size cells are not only easy to switch using a fast parallel fabric, but also easy to schedule. In contrast, IP, with its variable-length packets and need for a longest prefix match, has been much harder to route. This relative difficulty is reflected in the relative prices of ATM switches and IP routers: IP routers are about an order of magnitude more expensive per switched megabit per second of bandwidth. Advances in route lookup technology, however, have given IP a critical edge over ATM. Using the fast route lookup algorithms described here, routers can look up a longest prefix match almost as fast, and almost as cheaply, as VCIs. Moreover, by fragmenting IP packets into fixed-size units at input ports and reassembling them at output ports, ATM-like switching fabrics can be used even in IP routers. Finally, advances in scheduling have reduced the cost of complex scheduling algorithms sufficiently that the overhead for scheduling variable-size packets has decreased. Thus, in the near future the cost per switched megabit per second of IP bandwidth is likely to be only marginally higher than that of ATM. With this decrease, and the already dominant role of IP at the endpoints and enterprise networks, the role of ATM is rapidly diminishing. It is almost certain that ATM will play only a limited role in future networks, primarily as a data link technology for telephony, and perhaps as a bearer technology for ADSL.

increase the bandwidth available from each home. This will further increase the load on access routers. Because of these four trends, access routers will soon need to support a large number of heterogeneous, potentially high-speed ports and a variety of protocols operating at each port, and try to bypass the voice switch if this is at all possible. We feel that access router technology is in a state of flux due to recent advances in technologies such as ADSL, and it is too early to discuss their architecture in this article.

Recent Advances and Current Trends

The rapid growth of the Internet has led to a flurry of new research in router designs. In this section, we present a selection of these approaches, along with our observations about current trends in router design. A note of warning: this is not meant to be an exhaustive list—it is only a sampling of work in a rapidly changing field!

High-Speed Route Lookup

We saw previously that one of the major bottlenecks in backbone routers is the need to compute the longest prefix match for each incoming packet. The speed of a route lookup algorithm is determined by the number of memory accesses it requires to find the matching route entry, and the speed of the memory. For example, if an algorithm performs eight memory lookups and the input port has a memory with an access time of 60 ns, the time taken to look up a route is 480 ns, allowing it to do about 2 million route lookups/s. The same algorithm, using a costlier memory with a 10 ns access time, would allow the port to perform 12.5 million lookups/s. A second consideration in designing forwarding table data structures is the time taken to update the table. Recent studies have shown that a routing table changes relatively slowly, requiring updates only around once every 2 min [7]. This allows us to use complicated data structures that optimize route lookup at the expense of the time taken to update the routing table.

The standard data structure to store routes is a tree, where each path in the tree from root to leaf corresponds to an entry in the forwarding table. Thus, the longest prefix match is the longest path in the tree that matches the destination address of an incoming packet [8]. Conceptually, a tree-based algorithm starts at the root of the tree and recursively matches the children of the current node with the next few bits of the destination address, stopping if no match is found [9]. Thus, in the worst case it takes time proportional to the length of the destination address to find the longest prefix match. The key idea in a tree-based algorithm is that most nodes require storage for only a few children instead of all possible ones. Such algorithms, therefore, make frugal use of memory at the expense of doing more memory lookups. As memory prices drop, this is precisely the wrong design decision. Worse, the commonly used Patricia-tree algorithm may need to backtrack to find the longest match, leading to poor worst-case performance.

The performance of route lookup algorithms can be improved in several ways.[1] We classify these improvement techniques into:

- Hardware-oriented techniques
- Table compaction techniques
- Hashing techniques

[1] *These improvements are commercially very valuable, so they are not well described in the literature. Our descriptions are meant only as an outline; nondisclosure agreements preclude greater detail.*

Well-known hardware-oriented solutions are based on content-addressable memories (CAMs) and caches. Both techniques scale poorly with routing table size, and cannot be used for backbone routers that support large routing tables. Some recent hardware-oriented approaches essentially combine logic and memory together in a single device, drastically reducing the memory access time. This "intelligent-memory" approach is quite general, and can be used in conjunction with the software techniques described later. A second hardware-oriented solution is to increase the amount of memory used to store the routing table. Reference [10] argues that it is feasible to use a single 1 Gb table to look up a 32-bit address. Even at current prices, this would cost only about $6500, which is well within the range of affordability for backbone routers. Over time, as memory costs drop, this approach might well be the best one even for enterprise routers. A subtle problem with this approach, however, is that the table becomes very hard to update: changing a single forwarding entry might cause several thousand memory locations to be updated. Cheap special-purpose hardware that can perform rapid updates on the forwarding table is described in [10].

Table compaction techniques, such as the algorithm described in [11], exploit the sparse distribution of forwarding entries in the space of all possible network addresses to build a complicated but compact data structure for the forwarding table. The table is then stored in the primary cache of a processor, allowing route lookup at gigabit speeds.

Finally, hash-based solutions have also been proposed for route lookup. The need to determine the longest prefix match limits the use of hashing. In particular, given a destination address, we do not know the prefix to use for finding the longest match. The solution to this problem is to try different masks, choosing the one that has the longest mask length. The choice of masks can be iterative [12] or hierarchical, or the first few bits of the address could be used to find a list of prefix lengths. Unfortunately, none of these solutions scale well with the size of the destination address.

In recent work, Waldvogel *et al.* [13] have presented a scalable hash-based algorithm that can look up the longest prefix match for an N bit address in $O(\log N)$ steps. Their algorithm computes a separate hash table for each possible prefix length. Instead of naively searching for a successful hash starting from the longest possible prefix, their algorithm does a binary search on the prefix lengths. This requires hash tables to contain markers that, on a hash failure, point to the correct smaller-length hash table to search in. The search path for a particular forwarding entry is compactly stored in the form of a "rope," which reduces the storage requirements for markers. In addition, by precomputing hash tables that hold all forwarding entries associated with each 16-bit prefix, they can "mutate" the hash table on the fly, further reducing the number of memory accesses to an average of two per lookup.

To sum up, we believe that fast route lookup is a solved problem. This has major implications for ATM switches (see sidebar).

Advances in Switching Fabrics

As mentioned in the first section, switching fabrics are usually implemented as a crossbar, shared memory, or bus. The speed of a crossbar fabric is limited by the scheduler, that of a shared memory fabric by memory access speeds, and that of a bus by bus capacitance and arbitration overhead. The mid-'80s saw a lot of research in the area of switch fabric design, but few of these designs were actually built because advances in bus speeds made them unnecessary. These designs include the well-known Banyan family of fabrics, along with others such as the Delta and Omega fabrics (for a survey see [1]). These

designs were revived in the early '90s, mostly for building large ATM switches. With the recent decline in demand for ATM, IP routers are being built by wrapping segmentation and reassembly modules around ATM switch fabric cores. In these routers, permanent virtual circuits are established from each port to all the other ports. After a longest-prefix match determines the destination port for an IP packet, it is fragmented into ATM cells and switched. The ATM cells are reassembled at the output port before transmission. Using an ATM core allows the router to support different QoS streams in the switching fabric and overcome the problems associated with switching variable-size packets. However, these designs inherit some of drawbacks of ATM. First, ATM switches, for the most part, do not have good support for multicast.[2] Multicasting data through the switch core requires a VCI to be mapped to multiple VCIs, and copies of a cell to be generated either at the input port or within the switch fabric. These overheads decrease the efficiency of the switching fabric. Second, a subtler problem arises because IP traffic control algorithms are usually specified in terms of packets rather than in terms of cells. Thus, with cell-based fabrics, implementing semantics such as those required by shared filters in RSVP can be a challenging task. Despite these drawbacks, the use of a fixed-size internal switching core seems to be a widespread design technique, and we will assume the existence of such a core in the rest of this article.

Speeding Up Output Queues

We noted earlier that a major problem in output-queued switches is the speed at which the output queue can be accessed. Two design techniques allow this bottleneck to be overcome. The first is to build very wide memories that can load an entire cell in a single memory cycle. We can do this by deploying memory elements in parallel and feeding them with a cell-wide data bus. Although this extravagant use of memory is costly, as memory prices continue to drop by 60 percent a year the approach is rapidly becoming attractive. A second approach to building fast output queues is to integrate a port controller and the associated queues on a single chip. This approach allows the read/write control logic to access queues in memory an entire row at a time, and therefore at speeds far greater than with external logic. An example of this approach can be seen in the 77V400 chipset from IDT Inc. The key idea here is to integrate eight serial input and output port controllers and a shared memory on a single very large-scale integrated (VLSI) chip. The serial inputs are parallelized in a shift register, and the entire shift register, usually containing an ATM cell, is read into the memory in parallel. When the output port scheduler decides to serve a packet, it reads the cell in parallel into a shift register, converts it to serial, and transmits it. Switching is accomplished by deciding which output port controller should receive an incoming cell. Since the memory can be accessed rapidly, an output port controller can store eight cells in a single cell time (at a line rate of 155 Mb/s). The VLSI packaging makes the chip economical: a 1.2 Gb/s 8-port switch-on-a-chip costs only around $50. While this approach does not scale to very large switches, 77V400 chips can be interconnected to form larger buffered switch fabrics. For instance, a three-stage buffered Banyan switch can be easily constructed with such elements. Again, at each stage, cell loss due to simultaneous cell arrivals on multiple inputs is avoided because of the integration of the memory element with the port logic. We believe that intelligent port controllers, such as IDT's 77V400, Lucent's Atlanta, and MMC's ATMS2000 chipsets, are the wave of the future.

[2] *Currently multicast is not widely supported in the backbone. Efforts like the IP Multicast Initiative (IPMI) may change this in the near future, making multicast support an important feature for backbone routers.*

Input-Queued Switches

Despite improvements in the speed of output queues, they are still a significant bottleneck, since, to avoid packet loss, they must run much faster than the input links. We can avoid this problem altogether by building input-queued switches. Input queuing is often deprecated because of the head-of-line blocking problem: packets blocked at the head of an input queue prevent schedulable packets deeper within the queue from accessing the switch fabric. However, by maintaining per-output queues at each input, head-of-line blocking can be completely avoided. This still leaves the problem of arbitrating access to the switch fabric at high speeds. Recent research suggests that this may be solvable with current technology [5]. Consequently, it appears that router bandwidth can be increased by yet another order of magnitude by moving to input queuing. Input queuing, however, suffers from some serious problems that still need resolution. First, packet scheduling algorithms for providing QoS are usually specified in terms of output queues. It is not clear how to modify these algorithms to simultaneously schedule the output queues and the switch fabric. This is a complex and nontrivial issue: in effect, we are asking each input port controller to mimic the actions of the entire set of output port controllers, each of which could conceivably be transmitting packets on a different link technology. For example, consider a router that has a fiber distributed data interface (FDDI), a Fast Ethernet, and a T3 port. If this router uses input queuing, each input port controller should schedule packets not only according to the varying transmission speeds of the output links, but also in accordance with the transmission policies associated with the disparate links. In particular, an input queue cannot send a packet to the Fast Ethernet port if that port is backing off from a collision. It cannot send a packet to the FDDI port if that port does not have the token. Clearly, with the diversity of link technologies, building a general-purpose input port controller is a challenging if not impossible task. Second, enhanced router services such as Random Early Discard [14] depend on the length of the output queue. With an input-queued switch, the output queue length is not known. Due to these practical problems, nontrivial input-queued enterprise routers that deal with heterogeneous links and policies may never become practical, and hybrid approaches with both input and output queuing and a moderate degree of speedup in the switching fabric may be necessary.

Scheduling

Suppose that packets arriving at all the input ports of a router wish to leave from the same output port. If the output trunk speed is the same as the input trunk speed, only *one* of these packets can be transmitted in the time it takes for *all* of them to arrive at the output port. In order to prevent packet loss, the output port provides buffers to store excess arriving packets, and serves packets from the buffer as and when the output trunk is free. The obvious way to serve packets from the buffer is in the order in which they arrived at the buffer, that is, first-come first-served (FCFS). FCFS service is trivial to implement, requiring the router or switch to store only a single head and tail pointer per output trunk. However, this solution has its problems because it does not allow the router to give some sources a lower delay than others, or prevent a malicious source, which sends an unending

stream of packets as fast as it can, from causing other well-behaved streams from losing packets. An alternative service method, called Fair Queuing, solves these problems, albeit at a greater implementation cost [15]. In the Fair Queuing approach, each source sharing a bottleneck link is allocated an ideal rate of service at that link. Specifically, focusing only on the sources that are backlogged at the link at a given instant in time, the available service rate of the trunk is partitioned in accordance with a set of weights. If we represent the weights of backlogged sources 1, 2, ..., n by w_1, w_2,..., w_n, then the ideal service received by source k is $rw_k/\Sigma(w_i)$ where r is the rate of the outgoing trunk. Fair Queuing and its variants are mechanisms that serve packets from the output queue to approximately partition the trunk service rate in this manner. All versions of Fair Queuing require packets to be served in an order different from that in which they arrived. Consequently, Fair Queuing is more expensive to implement than FCFS, since it must decide the order in which to serve incoming packets, and then manage the queues in order to carry this out. In general, the higher the number of conversations going through a router, the costlier it is to implement Fair Queuing since Fair Queuing requires some form of per-conversation state to be stored on the routers. An exhaustive survey of variants of the Fair Queuing algorithm can be found in [16].

Fair Queuing has three important and useful properties. First, it provides *protection* so that a well-behaved source does not see packet losses due to misbehavior by other sources. Second, by design it provides fair bandwidth allocation. If the sum of weights of the sources is bounded, each source is guaranteed a minimum share of link capacity. Finally, it can be shown that if a source is leaky-bucket regulated, independent of the behavior of the other sources, it receives a bound on its worst-case end-to-end delay. For these reasons, almost all current routers support some variant of Fair Queuing.

A related scheduling problem has to do with the partitioning of link capacity among different classes of users. Consider a wide-area trunk shared by two companies. All other things being equal, in times of congestion the trunk should be equally shared by packets from both companies. These sort of link-sharing requirements deal with classes of connections rather than individual connections, and require per-class bookkeeping. In recent work it has been shown that extensions of Fair Queuing are compatible with hierarchical link-sharing requirements [17, 18]. Fast implementations of algorithms that provide both hierarchical link sharing and per-connection QoS guarantees are an area of active research [19, 20]. We expect that all future routers will provide some form of Fair Queuing at output queues.

Reducing Port Cost

Both enterprise and access routers support a large number of ports. Thus, they need to reduce the cost of a port to the bare minimum. The cost of a port depends on:

- The amount and kind of memory it uses
- Its processing power
- The complexity of the protocol used for communication between the port and the routing processor

Ports built with general-purpose processors, large buffers, and complex communication protocols tend to be more expensive than those built using ASICs, with smaller buffers and simple communication protocols. Choosing between ASICs and general-purpose processors for a port is not straightforward. General-purpose processors are costlier, but allow extensible port functionality, are available off-the-shelf, and their price/performance ratio improves rapidly with time. Their cost currently makes them suitable only for backbone routers, but their flexibility will eventually make them attractive for enterprise and access routers. On the other hand, ASICs are not only cheaper, but can also provide operations that are specific to routing, such as traversing a Patricia tree. Moreover, the lack of flexibility with ASICs can be overcome by implementing functionality in the routing processor. Thus, at the moment it seems to make sense to use processor-based designs for backbone routers and ASIC-based designs for the local area. Over time, the situation may well be reversed.

The cost of a port is proportional to the type and size of memory on the port. SRAMs offer faster access times, but are costlier than DRAMs. In general, backbone routers use SRAMs, and enterprise and access routers use DRAMs. Buffer memory is another parameter that is difficult to size. In general, the rule of thumb is that a port should have enough buffers to support at least one bandwidth delay product worth of packets, where the delay is the mean end-to-end delay and the bandwidth is the largest bandwidth available to TCP connections traversing that router. This sizing allows TCP connections to open up their transmission windows without excessive congestive losses. The largest bandwidth available to connections in the Internet today is around 100 Mb/s. In the backbone, assuming conservatively that the mean connection delay is 100 ms, this comes to about 1.125 Mb of buffering per port. Far less buffering is necessary in enterprise networks, where the mean connection delay is usually less than 10 ms, corresponding to a per-port buffering of about 100 kb.

Finally, the cost of the port is also determined by the complexity of the connections between the control path and the data path in the line card. In some cases, a routing processor sends commands to each port through the switching fabric and the port's internal buffers. If command packets can get lost they need retransmission. Careful engineering of the control protocol is necessary to reduce the cost of port control.

Enterprise-Level Management and Centralization

An enterprise or campus may contain many routers under the control of a single administrator. In this situation, it is often a good idea to centralize some functions usually associated with individual routers. For example, a central route server can compute loop-free routes for the entire enterprise and load them into each router's forwarding tables. A similar approach can be taken for loading multicast forwarding entries, thus freeing the routers from the burden of participating in complex multicast routing protocols. We call this "enterprise-level management." The enterprise-level approach also makes it easier to implement global policies. For example, an organization may want to limit the total amount of resources dedicated to multicast traffic. Although the mechanisms to restrict traffic (like policers) may be implemented at each router, the computation of parameters for the individual policers can be centralized.

Avoiding Route Lookups

Instead of reducing the cost of route lookups, backbone routers can use two techniques to avoid route lookups altogether. First, backbone networks can provide a virtual circuit interface (e.g., carrying IP over ATM over SONET), and require edge networks to translate from the destination address to a VCI. Since VCIs are integers drawn from a small space, they can be looked up with a single memory access. Moreover, the complexity of a longest prefix match is avoided. However, the edge network must somehow distribute the mapping between a VCI (or tag) and the destination port to

each router in the backbone. This can be achieved through protocols such as IP switching[3] and tag switching. The deployment of such a technique requires a major change to the network. A second technique essentially introduces another level in the routing hierarchy to reduce the size of routing tables, making route lookups cheaper. With this approach, backbone routers only keep routes to destinations served by that backbone. All unknown destinations are routed to a "gateway" at a network access point (NAP). Competing backbone providers exchange routing information and packets at NAPs. While this means that global (large) routing tables are needed only at NAPs, the growth in the number of providers in the NAP has stressed the scalability of the Border Gateway Protocol (BGP), which is used to exchange routing information among peer routers at the NAP.

Router Operating Systems

In the past, routers have been viewed as hardware devices that are optimized for routing packets at high speeds. Thus, the software environment on a typical control processor is bare bones, providing little beyond a basic monitor. There is a growing realization, however, that router hardware is getting to be a commodity which is easy to build, and the greatest asset of a router vendor is its software. A parallel trend, which reinforces this idea, is the notion that by opening up the router architecture to third parties, router vendors can leverage them to create enhanced services within the network. Thus, there is an intense focus on the development of router operating systems: operating systems that are specialized to run on routers and provide a carefully controlled application programming interface (API) to the underlying hardware. If this trend continues, end users may be able to install custom software modules within routers to provide services such as firewalls, traffic management policies, application-specific signaling, and fine-grained control of the routing policy. There is a fairly large and vocal "open signaling" community that is lobbying router vendors for precisely this kind of access to router internals [21]. It is interesting to note that telephone companies already provide an extremely rich programming interface to PBX hardware through the JTAPI interface [22]. An extreme variant of this trend, dubbed "active networking," would allow individual packets to install software on the fly in routers. It is not clear what performance penalty such an approach would exact, but recent work shows that both the security concerns and the performance hit of active networks may be tolerable, at least in some environments [23].

Open Problems

In this section, we identify some challenging open problems in router design. We believe that the solutions to these problems will lead to interesting new trade-offs in the next generation of IP routers.

Flow Identification

It is often useful to think of the set of packets traveling through the Internet between a given source and a given destination close together in time as constituting a flow. A flow can result from the set of packets within a long-lasting TCP connection or from the set of UDP packets in an audio or video session. By definition flows last for a while, so it is a useful optimization to pin resources, such as cache entries, associated with the current set of flows. Therefore, it is useful to identify flows on the fly by noticing, for instance, that more than X packets with the same source and destination IP addresses and TCP port numbers have been seen in the last Y seconds [24]. Flows may also be associated with real-time performance guarantees. We can identify these flows by matching incoming packet headers with a set of prespecified filters. Since classification needs to be done for each incoming packet, we need fast classification algorithms. Unfortunately, while the algorithms described earlier can look up routes for 32-bit addresses at line speed, they cannot be easily modified for fast flow classification. Moreover, we lack generic yet efficient flow descriptors. For instance, the most generic classifier is one that masks the source and destination IP addresses and ports and the protocol number, thus requiring a lookup on 104 bits of the packet. This sort of classifier seems difficult to implement at high speed. We believe that coming up with a concise description of a classifier and a way to match the "best" classifier among the several thousand that may be present at a router is an open problem.

Resource Reservation

The Internet has poor support for resource reservations: Ethernet-based LANs, WAN access links, and backbone routers are geared toward best-effort traffic, with no support even for the simplest of priority schemes. As Ethernets become switched and the demand for some form of service quality increases, we expect to see support for resource reservation in all three classes of routers. Resource reservation goes hand in hand with flow classification, because resources are reserved on behalf of prespecified flows. Unfortunately, this coupling makes resource reservation at least as hard to solve as this open problem!

Even if we had efficient flow classifiers, resource reservation additionally requires either policing, so the demand of an individual flow is limited, or some form of segregation in packet scheduling, so over-limit flows are automatically discouraged. Given the complexity of implementing Fair-Queuing-type scheduling algorithms at high speed, there has been much recent work in coming up with efficient policers. For example, in the RIO scheme over-limit packets are marked as low priority and preferentially discarded [25]. The choice of good policing algorithms and associated pricing schemes is an open problem.

Ease of Configuration

Configuring routers is hard work. Misconfigured routers can be hard to detect and can cause nearly untraceable performance problems. For example, bugs in the configuration of proxy ARP on routers manifest themselves only as a mysterious increase in network delay [26]. We believe that simple and intuitive abstractions of the underlying network functionality would go a long way in solving these problems. These abstractions remain elusive. Configuration becomes harder if the functionality, such as limiting the amount of multicast traffic in a network, requires the simultaneous configuration of more than one router in the network. Interaction between inconsistent configurations can cause networkwide problems and failures. It is not always possible to visually examine configuration files to discover mistakes and inconsistencies. We believe that the next generation of configuration tools will need rule-based and simulation-based subsystems to test a configured router before installing it in the field. This is a difficult and interesting open problem.

[3] *In IP switching, the backbone tells the edge about the mapping, instead of the other way around, but the idea is conceptually similar to what we describe here.*

Stability of Large Systems

Recall that router hardware can be made more reliable by adding hot spares, dual power supplies, and duplicate data paths. In contrast to hardware reliability, which is a well understood and solved problem, software reliability remains a challenging open problem. We believe that stability of router software is a necessary prerequisite for the reliability of a large network. Software stability is hard to achieve because software state is affected by interaction among different features. For example, the addition of a BGP (exterior routing) attribute may affect the calculation of routes exported to interior routing protocols. Furthermore, interaction between bugs from different vendors [7] can lead to persistent instabilities in the network. Simulation may not be very helpful since it is difficult to reproduce bugs in implementations, and thus the exact behavior of nodes in the network. We believe the one solution to software reliability may lie in adding features to protocol implementations, similar to the support for multicast traceroute in mrouted, which allow us to detect and isolate problems. This would at least allow us to track the cause for instability once problems occur.

Accountability

The introduction of differential service in the Internet must necessarily be accompanied by pricing. Pricing requires router support for accounting. The cost and feasibility of accounting support depends on the granularity at which accounting is done. Similar to flow identification, where coming up with a concise definition of a classifier and a way to match the best classifier is hard, a concise definition of an account and a way to identify and bill an account is an open problem.

Conclusions

IP routers are in the midst of great change, due to both customer pull and technology push. Customers are demanding higher bandwidth, greater reliability, lower cost, greater flexibility, and ease of configuration. Simultaneously, technology, in the form of ATM switching cores and fast route-lookup algorithms, has allowed router vendors to build the next generation of routers. We believe that the advances described in this article, such as the use of ATM cores, better output queuing, advanced scheduling algorithms, avoiding route lookups, and centralized administration, will be the distinguishing features of this generation. While these advances have solved some difficult problems, important issues still remain unresolved. We believe that understanding the stability of a network of routers is a critical open issue. Trading off cost, speed, flexibility, and ease of configuration, as always, will be a challenge for router designers in years to come.

Acknowledgments

We would like to thank Hemant Kanakia and Ritesh Ahuja for helping us better understand the "real" problems and trends in router design.

References

[1] Y. Oie *et al.*, "Survey of Switching Techniques in High-Speed Networks and Their Performance," *Proc. IEEE INFOCOM '90*, June 1990, pp. 1242–51.
[2] S. Keshav, *An Engineering Approach to Computer Networking*, Reading, MA: Addison-Wesley, 1997.
[3] "Contribution of Packet Sizes to Packet and Byte Volumes," MCI vBNS http://www.nlanr.net/NA/Learn/ packetsize.html.
[4] "Flow Counts as a Function of Flow Granularity," FIX-West, http://www.nlanr.ner/NA/Learn/aggregation.html.
[5] N. McKeown, "Scheduling Algorithms for Input-Queued Cell Switches," Ph.D. thesis, Univ. Calif. Berkeley, May 1995.
[6] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *Proc. IEEE INFOCOM '96*, San Francisco, CA, Mar. 1996.
[7] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet Routing Instability," *Proc. ACM SIGCOMM '97*, Cannes, France, Sept. 1997.
[8] W. R. Stevens, *TCP/IP Illustrated*, vol. 2, Reading, MA: Addison-Wesley, 1995.
[9] K. Sklower, *A Tree-Based Packet Routing Scheme for Berkeley Unix*, Internal memo, Univ. Calif. Berkeley.
[10] P. Gupta, S. Lin and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," *Proc. IEEE INFOCOM '98*, Mar. 1998.
[11] A. Brodnik *et al.*, "Small Forwarding Tables for Fast Route Lookups," *Proc. ACM SIGCOMM '97*, Cannes, France, Sept. 1997.
[12] Online Proceeding of the panel discussion on Better Integration of IP and ATM, IP ATM Workshop, Washington University, St. Louis, MO, Nov. 1996; http://arl.wustl.edu/arl.
[13] M. Waldvogel *et al.*, "Scalable High Speed IP Routing Lookups," *Proc. ACM SIGCOMM '97*, Cannes, France, Sept. 1997.
[14] S. Floyd and V. Jacobson, Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Trans. Networking*, Aug. 1993.
[15] A. Demers, S. Keshav, and S. Shenker, "Design and Analysis of a Fair Queuing Algorithm," *Proc. ACM SIGCOMM '89*, Austin, TX, Sept. 1989.
[16] P. Goyal, "Packet Scheduling Algorithms for Integrated Service Networks," Ph.D. thesis, Univ. Texas, Austin, 1997.
[17] J. C. R. Bennett and H. Zhang, "Hierarchical Packet Fair Queuing Algorithms," *Proc. ACM SIGCOMM '96*, Stanford, CA, Aug. 1996.
[18] P. Goyal, H. Vin, and H. Chen, "Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *Proc. ACM SIGCOMM '96*, Aug. 1996.
[19] J. C. R. Bennett, D. C. Stephens and H. Zhang, "High Speed, Scalable, and Accurate Implementation of Fair Queuing Algorithms in ATM Networks," *Proc. ICNP '97*, 1997.
[20] J. Rexford *et al.*, "A Scalable Architecture for Fair Leaky-Bucket Shaping," *Proc. IEEE INFOCOM '97*, Kobe, Japan.
[21] OPENSIG Spring '97 Workshop, http://www.ctr.columbia.edu/opensig
[22] Java Telephony Application Programmer's Interface, available from http://java.sun.com/products/jtapi/index.html
[23] D. S. Alexander *et al.*, "Active Bridging," *Proc. ACM SIGCOMM '97*, Cannes, France, Sept. 1997.
[24] S. Lin and N. McKeown, "A Simulation Study of IP Switching," *Proc. ACM SIGCOMM '97*, Cannes, France, Sept. 1997.
[25] D. Clark and J. Wroclawski, "An Approach to Service Allocations in the Internet," Internet draft, draft-clark-diff-svc- alloc-00.txt, available from http://www.ietf.org.
[26] S. M. Ballew, *Managing IP Networks with Cisco Routers*, O'Reilly, 1997.

Biographies

S. Keshav (skeshav@cs.cornell.edu) is an associate professor of computer science at Cornell University. He was formerly a member of technical staff at AT&T Bell Laboratories, where he worked on ATM network design and traffic management. He is the author of *An Engineering Approach to Computer Networks* (Addison-Wesley Longman, 1997). He received the Sakrison Prize for the best Ph.D. thesis in the EECS Department at University of California, Berkeley, in 1992, and a Sloan Fellowship in 1997. He is a member of the editorial teams for *IEEE/ACM Transactions on Networking* and the *Journal of High Speed Networks*, and has published numerous technical papers. His current research interests are in network management, protocol design, and computer telephony integration.

Rosen Sharma completed his B.Tech in computer Science from the Indian Institute of Technology, Delhi, in 1993, receiving the President's Gold Medal for academic excellence. He then joined Stanford University to pursue a doctoral degree in networking. His research led to the formation of VxTreme Corporation in December 1995 in Palo Alto, California, of which he was cofounder and lead architect. VxTreme developed client/server applications for multimedia delivery on the Internet. His is currently a Ph.D. candidate at Cornell University. His research interests are in the areas of multimedia and networking.