IP Router Architectures: An Overview

by

James Aweya

{Email: aweyaj@nortelnetworks.com; Tel: 1-613-763-6491; Fax: 1-613-763-5692}

Nortel Networks

Ottawa, Canada, K1Y 4H7

Abstract

In the emerging environment of high performance IP networks, it is expected that local and campus area backbones, enterprise networks, and Internet Service Providers (ISPs) will use multigigabit and terabit networking technologies where IP routers will be used not only to interconnect backbone segments but also to act as points of attachments to high performance wide area links. Special attention must be given to new powerful architectures for routers in order to play that demanding role. In this paper, we identify important trends in router design and outline some design issues facing the next generation of routers. It is also observed that the achievement of high throughput IP routers is possible if the critical tasks are identified and special purpose modules are properly tailored to perform them.

1. Introduction

The popularity of the Internet has caused the traffic on the Internet to grow drastically every year for the last several years. It has also spurred the emergence of many Internet Service Providers (ISPs). To sustain growth, ISPs need to provide new differentiated services, e.g., tiered service, support for multimedia applications, etc. The routers in the ISPs' networks play a critical role in providing these services. Internet Protocol (IP) traffic on private enterprise networks has also been growing rapidly for some time. These networks face significant bandwidth challenges as new application types, especially desktop applications uniting voice, video, and data traffic need to be delivered on the network infrastructure. This growth in IP traffic is beginning to stress the traditional processor-based design of current-day routers and as a result has created new challenges for router design.

Routers have traditionally been implemented purely in software. Because of the software implementation, the performance of a router was limited by the performance of the processor executing the protocol code. To achieve wire-speed routing, high-performance processors together with large memories were required. This translated into higher cost. Thus, while software-based wire-speed routing was possible at low-speeds, for example, with 10 megabits per second (Mbps) ports, or with a relatively smaller number of 100 Mbps ports, the processing costs and architectural implications make it difficult to achieve wire-speed routing at higher speeds using software-based processing.

Fortunately, many changes in technology (both networking and silicon) have changed the landscape for implementing high-speed routers. Silicon capability has improved to the point where highly complex systems can be built on a single integrated circuit (IC). The use of 0.35 μm and smaller silicon geometries enables application specific integrated circuit (ASIC) implementations of millions gate-equivalents. Embedded memory (SRAM, DRAM) and microprocessors are available in addition to high-density logic. This makes it possible to build single-chip, low-cost routing solutions that incorporate both hardware and software as needed for best overall performance.

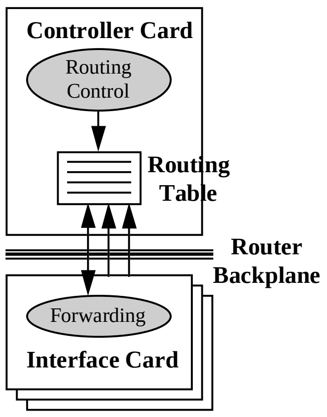
In this paper we investigate the evolution of IP router designs and highlight the major performance issues affecting IP routers. The need to build fast IP routers is being addressed in a variety of ways. We discuss these in various sections of the paper. We discuss in detail the various router mechanisms needed for high-speed operation. In particular, we examine the architectural constraints imposed by the various router design alternatives. The scope of the discussion presented here does not cover more recent *label switching* routing techniques such as IP Switching [1], the Cell Switching Router (CSR) architecture [2], Tag Switching [3], and Multiprotocol Label Switching (MPLS), which is a standardization effort underway at the Internet Engineering Task Force (IETF). The discussion is limited to routing techniques as described in RFC 1812 [4].

In Section 2, we briefly review the basic functionalities in IP routers. The IETF's *Requirements for IP Version 4 Routers* [4] describes in great detail the set of protocol standards that Internet Protocol version 4 (IPv4) routers need to conform to. Section 3 presents the design issues and trends that arise in IP routers. The most common switch fabric technologies in use today are buses, shared memories, and crossbars. Section 4 presents an overview of these switch fabric technologies. The concluding remarks are given in Section 5.

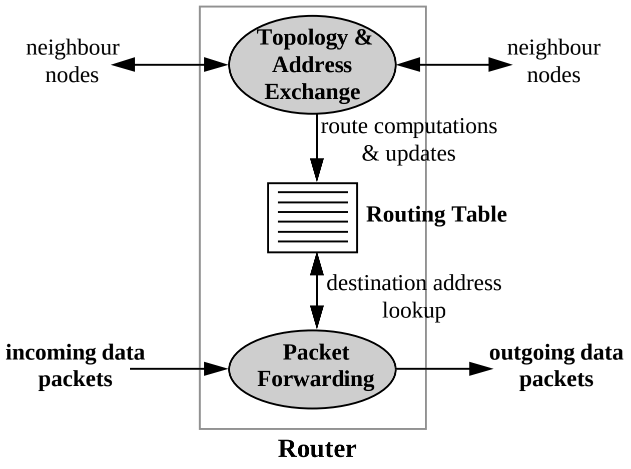
2. Basic IP Router Functionalities

Generally, routers consist of the following basic components: several network interfaces to the attached networks, processing module(s), buffering module(s), and an internal interconnection unit (or switch fabric). Typically, packets are received at an inbound network interface, processed by the processing module and, possibly, stored in the buffering module. Then, they are forwarded through the internal interconnection unit to the outbound interface that transmits them on the next hop on the journey to their final destination. The aggregate packet rate of all attached network interfaces needs to be processed, buffered and relayed. Therefore, the processing and memory modules may be replicated either fully or partially on the network interfaces to allow for concurrent operations.

A generic architecture of an IP router is given in Figure 1. Figure 1a shows the basic architecture of a typical router: the controller card (which holds the CPU), the router backplane, and interface cards. The CPU in the router typically performs such functions as path computations, routing table maintenance, and reachability propagation. It runs which ever routing protocols is needed in the router. The interface cards consists of adapters that perform inbound and outbound packet forwarding (and may even cache routing table entries or have extensive packet processing capabilities). The router backplane is responsible for transferring packets between the cards. The basic functionalities in an IP router can be categorized as: route processing, packet forwarding, and router special services. The two key functionalities are route processing (i.e., path computation, routing table maintenance, and reachability propagation) and packet forwarding (see Figure 1b). We discuss the three functionalities in more detail below.



a). Basic architecture.



b). Routing components.

Figure 1. Generic architecture of a router.

- *Route Processing*:

  This includes routing table construction and maintenance using routing protocols (such as RIP or OSPF) to learn about and create a view of the network's topology [5][6][7]. Updates to the routing table can also be done through management action where routes are added and deleted manually.

- *Packet Forwarding*:

  Typically, IP packet forwarding requires the following:

  - ♦ **IP Packet Validation:** The router must check that the received packet is properly formed for the protocol before it proceeds with protocol processing. This involves checking the version number, checking the header length field (also needed to determine whether any options are present in the packet), and calculating the header checksum.

♦ **Destination IP Address Parsing and Table Lookup:** The router performs a table lookup to determine the output port onto which to direct the packet and the next hop to which to send the packet along this route. This is based on the destination IP address in the received packet and the subnet mask(s) of the associated table entries. The result of this lookup could imply:

- A local delivery (that is, the destination address is one of the router's local addresses and the packet is locally delivered).
- A unicast delivery to a single output port, either to a next-hop router or to the ultimate destination station (in the case of a direct connection to the destination network).
- A multicast delivery to a set of output ports that depends on the router's knowledge of multicast group membership.

The router must also determine the mapping of the destination network address to the data link address for the output port (address resolution or ARP). This can be done either as a separate step or integrated as part of the routing lookup.

♦ **Packet Lifetime Control:** The router adjusts the time-to-live (TTL) field in the packet used to prevent packets from circulating endlessly throughout the internetwork. A packet being delivered to a local address within the router is acceptable if it has any positive value of TTL. A packet being routed to output ports has its TTL value decremented as appropriate and then is rechecked to determine if it has any life before it is actually forwarded. A packet whose lifetime is exceeded is discarded by the router (and may cause an error message to be generated to the original sender).

♦ **Checksum Calculation:** The IP header checksum must be recalculated due to the change in the TTL field. Fortunately, the checksum algorithm employed (a 16-bit one's complement addition of the header fields) is both commutative and associative, thereby allowing simple, differential recomputation. RFC 1071 [8] contains implementation techniques for computing the IP checksum. Since a router often changes only the TTL field (decrementing it by 1), a router can incrementally update the checksum when it forwards a received packet, instead of calculating the checksum over the entire IP header again. RFC 1141 [9] describes an efficient way to do this.

IP packets might also have to be fragmented to fit within the Maximum Transmission Unit (MTU) specified for the outgoing network interface. Fragmentation, however, can affect performance adversely [10] but now that IP MTU discovery is prevalent [11], fragmentation should be rare.

- *Special Services*:

  Anything beyond core routing functions falls into this category: packet translation, encapsulation, traffic prioritization, authentication, and access services such as packet filtering for security/firewall purposes. In addition, routers possess network management components (e.g., SNMP, Management Information Base (MIB), etc.).

2.1 Route Table Lookup

For a long time, the major performance bottleneck in IP routers has been the time it takes to look up a route in the routing table. The problem is defined as that of searching through a database (routing table) 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 for increasing the number of networks addressed through Classless Inter-Domain Routing (CIDR) [12]. The CIDR technique is used to summarize a block of class C addresses into a single routing table entry. This consolidation results in a reduction in the number of separate routing table entries.

Given a packet, the router performs a routing table lookup, using the packet's IP destination address as key. This lookup returns the best-matching routing table entry, which tells the router which interface to forward the packet out of and the IP address of the packet's next hop

The first approaches for longest prefix matching used radix trees or modified Patricia trees [13][14] combined with hash tables, (Patricia stands for “Practical Algorithm to Retrieve Information Coded in Alphanumeric”). These trees are binary trees, whereby the tree traversal depends on a sequence of single-bit comparisons in the key, the destination IP address. These lookup algorithms have complexity proportional to the number of address bits which, for IPv4 is only 32. In the worst case it takes time proportional to the length of the destination address to find the longest prefix match. Worse, the commonly used Patricia algorithm may need to backtrack to find the longest match, leading to poor worst-case performance. The performance of Patricia is somewhat data dependent. With a particularly unfortunate collection of prefixes in the routing table, the lookup of certain addresses can take as many as 32 bit comparisons, one for each bit of the destination IP address.

Early implementations of routers, however, could not afford such expensive computations. Thus, one way to speed up the routing table lookup is to try to avoid it entirely. The routing table lookup provides the next hop for a given IP destination. Some routers cache this IP destination-to-next-hop association in a separate database that is consulted (as the front end to the routing table) before the routing table lookup. Finding a particular destination in this database is easier because an exact match is done instead of the more expensive best-match operation of the routing table. So, most routers relied on route caches [15][16]. The route caching techniques rely on there being enough locality in the traffic so that the cache hit rate is sufficiently high and the cost of a routing lookup is amortized over several packets.

This front-end database might be organized as a *hash table* [17]. After the router has forwarded several packets, if the router sees any of these destinations again (a *cache hit*), their lookups will be very quick. Packets to new destinations will be slower, however, because the cost of a failed hash lookup will have to be added to the normal routing table lookup. Front-end caches to the routing table can work well at the edge of the Internet or within organizations. However, cache schemes do not seem to work well in the Internet’s core. The large number of packet destinations seen by the core routers can cause caches to overflow or for their lookup to become slower than the routing table lookup itself. Cache schemes are not really effective when the hash bucket size (the number of destinations that hash to the same value) starts getting large. Also, the frequent routing changes seen in the core routers can force them to invalidate their caches frequently, leading to a small number of cache hits.

Typically, two types of packets arrive at a router: packets to be forwarded to another network or packets destined to the router itself. Whether a packet to a router causes a routing table reference depends on the router implementation. Some implementations may speed up routing table lookups. One possibility is for the router to explicitly check each incoming packet against a table of all of the router's addresses to see if there is a match. This explicit check means that the routing table is never consulted about packets destined to the router. Another possibility is to use the routing table for all packets. Before the packet is sent, the router checks if the packet is to its own address on the appropriate network. If the packet is for the router, then it is never transmitted. The explicit check after the routing table lookup requires checking a smaller number of router addresses at the increased cost of a routing table lookup. New routing table lookup algorithms are still being developed in attempts to build even faster routers. Recent examples are found in the references [18][19][20][21][22][23].

The basic idea of one of the recent algorithms [18] is to create a small and compressed data structure that represents large lookup tables using a small amount of memory. The technique exploits the sparseness of actual entries in the space of all possible routing table entries. This results in a lower number of memory accesses (to a fast memory) and hence faster lookups. The proposal reduces the routing table to very efficient representation of a binary tree such that the majority of the table can reside in the primary cache of the processor, allowing route lookups at gigabit speeds. In addition, the algorithm does not have to calculate expensive perfect hash functions, although updates to the routing table are still not easy. Reference [19] proposes another approach for implementing the compression and minimizing the complexity of updates.

The recent work of Waldvogel et al. [20] presents an alternative approach which reduces the number of memory references rather than compact the routing table. 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 amongst prefixes of a particular length, can find the longest prefix match for an N bit address in $O(\log N)$ steps. Although the algorithm has very fast execution times, calculating perfect hashes can be slow and can slow down updates.

Hardware based techniques for route lookup are also actively being investigated both in research and commercial designs (e.g., [24][25][26][27]). Other designs of forwarding engine have concentrated on IP packet header processing in hardware, to remove the dependence upon caching, and to avoid the cost of high-speed processor. Designs based upon content-addressable memory have been investigated [24], but such memory is too far expensive to be applied to a large routing table. Hardware route lookup and forwarding is also under active investigation in both research and commercial designs [25][26]. The argument for a software-based implementation stresses flexibility. Hardware implementations can generally achieve a higher performance at lower cost but are less flexible.

3. IP Router Architectures

In the next sections, we examine important trends in IP router design and outline some design issues facing the next generation of routers.

3.1 Bus-based Router Architectures with Single Processor

The first generation of IP router were based on software implementations on a single general-purpose central processing unit (CPU). These routers consist of a general-purpose processor and multiple interface cards interconnected through a shared bus as depicted in Figure 2.

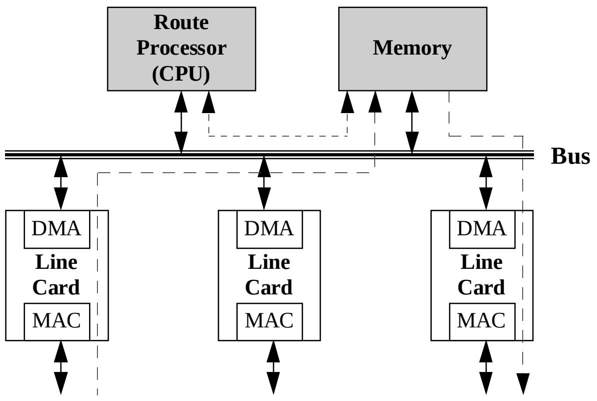


Figure 2. Traditional bus-based router architecture.

Packets arriving at the interfaces are forwarded to the CPU which determines the next hop address and sends them back to the appropriate outgoing interface(s). Data is usually buffered in a centralized data memory [28][29], which leads to the disadvantage of having the data cross the bus twice, making it the major system bottleneck. Packet processing and node management software (including routing protocol operation, routing table maintenance, routing table lookups, and other control and management protocols such as ICMP, SNMP) are also implemented on the central processor. Unfortunately, this simple architecture yields low performance for the following reasons:

- The central processor has to process all packets flowing through the router (as well as those destined to it). This represents a serious processing bottleneck.

- Some major packet processing tasks in a router involve memory intensive operations (e.g., table lookups) which limits the effectiveness of processor power upgrades in boosting the router packet processing throughput. Routing table lookups and data movements are the major consumer of processing cycles. The processing time of these tasks does not decrease linearly if faster processors are used. This is because of the sometimes dominating effect of memory access rate.

- Moving data from one interface to the other (either through main memory or not) is a time consuming operation that often exceeds the packet header processing time. In many cases, the computer input/output (I/O) bus quickly becomes a severe limiting factor to overall router throughput.

Since routing table lookup is a time-consuming process of packet forwarding, some traditional software-based routers cache the IP destination-to-next-hop association in a separate database that is consulted as the front end to the routing table before the routing table lookup [15]. Still, the performance of the traditional bus-based router depends heavily on the throughput of the shared bus and on the forwarding speed of the central processor. This architecture cannot scale to meet the increasing throughput requirements of multigigabit network interface cards.

3.2 Bus-based Router Architectures with Multiple Processors

3.2.1 Architectures with Route Caching

For the second generation IP routers, improvement in the shared-bus router architecture was introduced by distributing the packet forwarding operations. Distributing fast processors and route caches, in addition to receive and transmit buffers, over the network interface cards reduces the load on the system bus. Packets are therefore transmitted only once over the shared bus. This reduces the number of bus copies and speeds up packet forwarding by using a route cache of frequently seen addresses in the network interface as shown in Figure 3. This architecture allows the network interface cards to process packets locally some of the time.

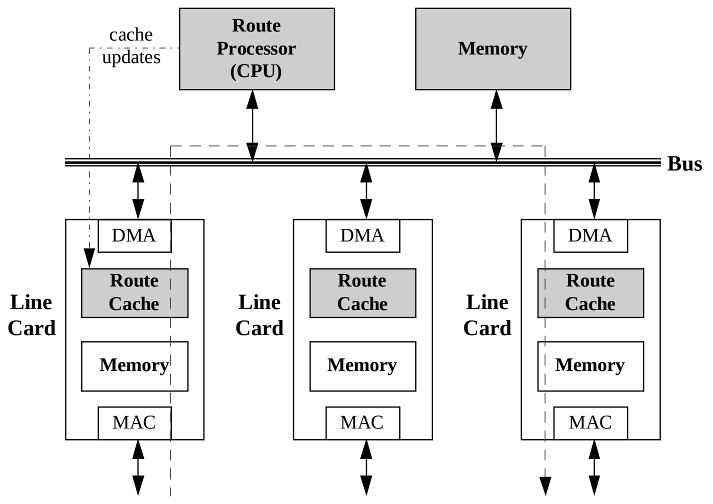


Figure 3. Reducing the number of bus copies using a route cache in the network interface.

In this architecture, a router keeps a central master routing table and the satellite processors in the network interfaces each keep only a modest cache of recently used routes. If a route is not in a network interface processor’s cache, it would request the relevant route from the central table. The route cache entries are traffic-driven in that the first packet to a new destination is routed by the main CPU (or route processor) via the central routing table information and as part of that forwarding operation, a route cache entry for that destination is then added in the network interface. This allows subsequent packet flows to the same destination network to be switched based on an efficient route cache match. These entries are periodically aged out to keep the route cache current and can be immediately invalidated if the network topology changes. At high-speeds, the central routing table can easily become a bottleneck because the cost of retrieving a route from the central table is many times more expensive than actually processing the packet local in the network interface.

A major limitation of this architecture is that it has a traffic dependent throughput and also the shared bus is still a bottleneck. The performance of this architecture can be improved by enhancing each of the distributed network interface cards with larger memories and complete forwarding tables. The decreasing cost of high bandwidth memories makes this possible. However, the shared bus and the general purpose CPU in the slow data path can neither scale to high capacity links nor provide traffic pattern independent throughput.

3.2.2 Architectures with Multiple Parallel Forwarding Engines

Another bus-based multiple processor router architecture is described in [30]. Multiple forwarding engines are connected in parallel to achieve high packet processing rates as shown in Figure 4. The network interface modules transmit and receive data from the links at the required rates. As a packet comes in, the IP header is stripped by the control circuitry, augmented with an identifying tag, and sent to a forwarding engine for validation and routing. While the forwarding engine is performing the routing function, the remainder of the packet is deposited in an input buffer in parallel. The forwarding engine determines which outgoing link the packet should be transmitted on, and sends the updated header fields to the appropriate destination interface module along with the tag information. The packet is then moved from the buffer in the source interface module to a buffer in the destination interface module and eventually transmitted on the outgoing link.

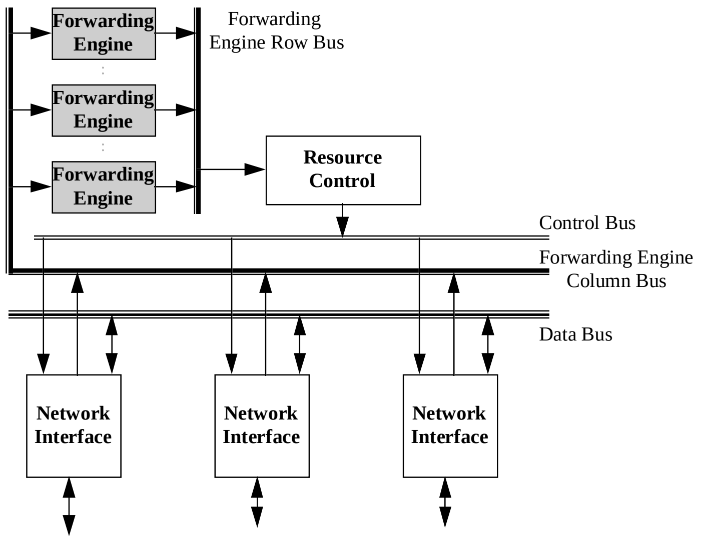


Figure 4. Bus-based router architecture with multiple parallel forwarding engines.

The forwarding engines can each work on different headers in parallel. The circuitry in the interface modules peels the header off of each packet and assigns the headers to the forwarding engines in a round-robin fashion. Since in some (real time) applications packet order maintenance is an issue, the output control circuitry also goes round-robin, guaranteeing that packets will then be sent out in the same order as they were received. Better load-balancing may be achieved by having a more intelligent input interface which assigns each header to the lightest loaded forwarding engine [30]. The output control circuitry would then have to select the next forwarding engine to obtain a processed header from by following the demultiplexing order followed at the input, so that order preservation of packets is ensured. The forwarding engine returns a new header (or multiple headers, if the packet is to be fragmented), along with routing information (i.e., the immediate destination of the packet). The route processor (controller) runs the routing protocols and creates a forwarding table that is used by the forwarding engines.

The choice of this architecture was premised on the observation 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. The forwarding engines are only responsible for resolving next-hop addresses. Forwarding only IP headers to the forwarding engines eliminates an unnecessary packet payload transfer over the bus. Packet payloads are always directly transferred between the interface modules and they never go to either the forwarding engines or the route processor unless they are specifically destined to them.

3.3 Switch-based Router Architectures with Multiple Processors

To alleviate the bottlenecks of the second generation of IP routers, the third generation of routers were designed with the shared bus replaced by a switch fabric. This provides sufficient bandwidth for transmitting packets between interface cards and allows throughput to be increased by several orders of magnitude. With the interconnection unit between interface cards not the bottleneck, the new bottleneck is packet processing.

The multigigabit router (MGR) is an example of this architecture [31]. The design has dedicated IP packet forwarding engines with route caches in them. The MGR consists of multiple line cards (each supporting one or more network interfaces) and forwarding engine cards, all connected to a high-speed (crossbar) switch as shown in Figure 5. The design places forwarding engines on boards distinct from line cards. When a packet arrives at a line card, its header is removed and passed through the switch to a forwarding engine. The remainder of the packet remains on the inbound line card. The forwarding engine reads the header to determine how to forward the packet and then updates the header and sends the updated header and its forwarding instructions back to the inbound line card. The inbound line card integrates the new header with the rest of the packet and sends the entire packet to the outbound line card for transmission. The MGR, like most routers, also has a control (and route) processor that provides basic management functions such as generation of routing tables for the forwarding engines and link (up/down) management. Each forwarding engine has a set of the forwarding tables (which are a summary of the routing table data).

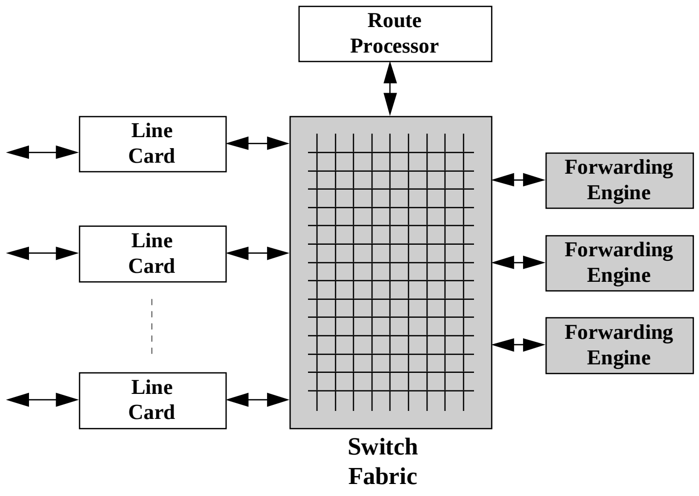


Figure 5. Switch-based router architecture with multiple forwarding engines.

In the MGR, once headers reach the forwarding engine, they are placed in a request first-in first-out (FIFO) queue for processing by the forwarding processor. The forwarding process can be roughly described by the following three stages [31].

1. The first stage includes the following which are done in parallel:

    - The forwarding engine does basic error checking to confirm that the header is indeed from an IPv4 datagram;
    - confirms that the packet and header lengths are reasonable;
    - confirms that the IPv4 header has no options;

- computes the hash offset into the route cache and loads the route; and
- starts loading the next header.

2. In the second stage, the forwarding engine checks to see if the cached route matches the destination of the datagram (a cache hit). If not, the forwarding engine carries out an extended lookup of the forwarding table associated with it. In this case, the processor searches the routing table for the correct route, and generates a version of the route for the route cache. Since the forwarding table contains prefix routes and the route cache is a cache of routes for particular destination, the processor has to convert the forwarding table entry into an appropriate destination-specific cache entry. Then, the forwarding engine checks the IP time-to-live (TTL) field and computes the updated TTL and IP checksum, and determines if the datagram is for the router itself.

3. In the third stage the updated TTL and checksum are put in the IP header. The necessary routing information is extracted from the forwarding table entry and the updated IP header is written out along with link-layer information from the forwarding table.

3.4 Limitation of IP Packet Forwarding based on Route Caching

Regardless of the type of interconnection unit used (bus, shared memory, crossbar, etc.), a route cache can be used in conjunction with a (centralized) processing unit for IP packet forwarding [15]. In this section, we examine the limitations of route caching techniques.

The route cache model creates the potential for cache misses which occur with "demand-caching" schemes as described above. That is, if a route is not found in the forwarding cache, the first packet(s) then looks to the routing table maintained by the CPU to determine the outbound interface and then a cache entry is added for that destination. This means when addresses are not found in the cache, the packet forwarding defaults to a classical software-based route lookup (sometimes described as a "slow-path"). Since the cache information is derived from the routing table, routing changes cause existing cache entries to be invalidated and reestablished to reflect any topology changes. In networking environments which frequently experience significant routing activity (such as in the Internet) this can cause traffic to be forwarded via the main CPU (the slow path), as opposed to via the route cache (the fast path).

In enterprise backbones or public networks, the combination of highly random traffic patterns and frequent topology changes tends to eliminate any benefits from the route cache, and performance is bounded by the speed of the software slow path which can be many orders of magnitude lower than the caching fast path.

This demand-caching scheme, maintaining a very fast access subnet of the routing topology information, is optimized for scenarios whereby the majority of the traffic flows are associated with a subnet of destinations. However, given that traffic profiles at the core of the Internet (and potentially within some large enterprise networks) do not follow closely this model, a new forwarding paradigm is required that would eliminate the increasing cache maintenance resulting from growing numbers of topologically dispersed destinations and dynamic network changes.

The performance of a product using the route cache technique is influenced by the following factors:

- how big the cache is,
- how the cache is maintained (the three most popular cache maintenance strategies are random replacement, first-in-first-out (FIFO), and least recent use (LRU)), and
- what the performance of the slow path is, since at least some percentage of the traffic will take the slow path in any application.

The main argument in favor of cache-based schemes is that a cache hit is at least less expensive than a full route lookup (so a cache is valuable provided it achieves a modest hit rate). Even with an increasing number of flows, it appears that packet bursts and temporal correlation in the packet arrivals will continue to ensure that there is a strong chance that two datagrams arriving close together will be headed for the same destination.

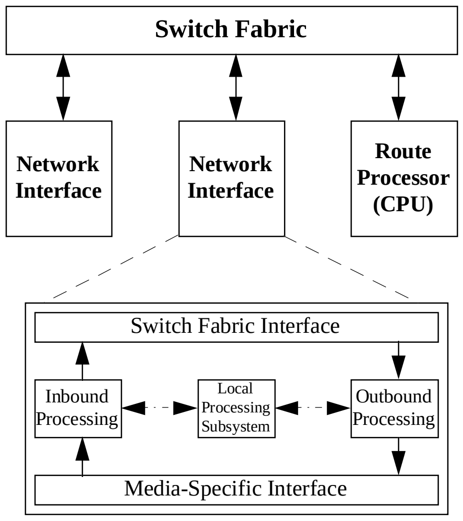
In current backbone routers, the number of flows that are active at a given interface can be extremely high. Studies have shown that an OC-3 interface might have an average of 256,000 flows active concurrently [32]. It is observed in [33] that, for this many flows, use of hardware caches is extremely 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. Due to the large number of flows that are simultaneously active in a router and due to 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 flows arrive at the same time, the slow path of the router can be overloaded, and it is possible that packet loss can occur due to the (slow path) processing overload and not due to output link congestion.

Some architectures have been proposed that avoid the potential overload of continuous cache churn (which results in a performance bottleneck) by instead using a forwarding database in each network interface which mirrors the entire content of the IP routing table maintained by the CPU (route processor), i.e., there is a one-to-one correspondence between the forwarding database entries and routing table prefixes; therefore no need to maintain a route cache [34][35][36]. By eliminating the route cache, the architecture fully eliminates the slow path. This offers significant benefits in terms of performance, scalability, network resilience and functionality, particularly in large complex networks with dynamic flows. These architectures can best accommodate the changing network dynamics and traffic characteristics resulting from increasing numbers of short flows typically associated with Web-based applications and interactive type sessions.

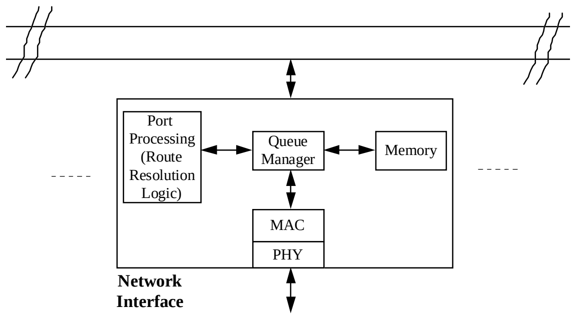
3.5 Switch-based Router Architectures with Fully Distributed Processors

From the discussion in the preceeding sections, we find that the three main bottlenecks in a router are: processing power, memory bandwidth, and internal bus bandwidth. These three bottlenecks can be avoided by using a distributed switch based architecture with properly designed network interfaces [34][35][36]. Since routers are mostly dedicated systems not running any specific application tasks, off-loading processing to the network interfaces reflects a proper approach to increase the overall router performance. A successful step towards building high performance routers is to add some processing power to each network interface in order to reduce the processing and memory bottlenecks. General-purpose processors and/or dedicated VLSI components can be applied. The third bottleneck (internal bus bandwidth) can be solved by using special mechanisms where the internal bus is in effect a switch (e.g., shared memory, crossbar, etc.) thus allowing simultaneous packet transfers between different pairs of network interfaces. This arrangement must also allow for efficient multicast capabilities.

We investigate in this section, decentralized router architectures where each network interface is equipped with appropriate processing power and buffer space. A generic modular switch-based router architecture is shown in Figure 6.



a). Functional diagram [34][35][36].



b). Generic architecture.

Figure 6. A generic switch-based distributed router architecture.

Each network interface provides the processing power and the buffer space needed for packet processing tasks related to all the packets flowing through it. Functional components (inbound, outbound, and local processing elements) process the inbound, outbound traffic and time-critical port processing tasks. They perform the processing of all protocol functions (in addition to quality of service (QoS) processing functions) that lie in the critical path of data flow. In order to provide QoS guarantees, a port may need to classify packets into predefined service classes. Depending on router implementation, a port may also need to run data-link level protocols or network-level protocols. The exact features of the processing components depend on the functional partitioning and implementation details. Concurrent operation among these components can be provided. The network interfaces are interconnected via a high performance switch that enables them to exchange data and control messages. In addition, a CPU is used to perform some centralized tasks. As a result, the overall processing and buffering capacity is distributed over the available interfaces and the CPU.

The Media-Specific Interface (MSI) performs all the functions of the physical layer and the Media Access Control (MAC) sublayer (in the case of the IEEE 802 protocol model). The Switch Fabric Interface (SFI) is responsible for preparing the packet for its journey across the switch fabric. The SFI may prepend an internal routing tag containing port of exit, the QoS priority, and drop priority, onto the packet.

To analyze the processing capabilities and to determine potential performance bottlenecks, the functions and components of a router and, especially, of all its processing subsystems have to be identified. Therefore, all protocols related to the task of a router need to be considered. In an IP router, the IP protocol itself as well as additional protocols, such as ICMP, ARP, RARP, BGP, etc. are required.

First, a distinction can be made between the processing tasks directly related to packets being forwarded through the router and those related to packets destined to the router, such as maintenance, management or error protocol data. Best performance can be achieved when packets are handled by multiple heterogeneous processing elements, where each element specializes in a specific operation. In such a configuration, special purpose modules perform the time critical tasks in order to achieve high throughput and low latency. Time critical tasks are the ones related to the regular data flow. The non-time critical tasks are performed in general purpose processors (CPU). A number of commercial routers follow this design approach (e.g., [33][37][38][39][40][41][42][43][44][45[46]).

3.5.1 Critical Data Path Processing (Fast Path)

The *time critical processing tasks* forms the *critical path* (sometimes called the *fast path*) through a router and need to be highly optimized in order to achieve multigigabit rates. These processing tasks comprise all protocols involved in the critical path (LLC, SNAP, and IP) as well as ARP which can be processed in the network interface because it needs direct access to the network, even though it is not time critical. The time critical tasks mainly consist of header checking, and forwarding (and may include segmentation) functions. These protocols directly affect the performance of an IP router in terms of the number of packets that can be processed per second.

The router architecture should be optimized for those fast path functions that must be performed in real time. Most high-speed routers implement this fast path in hardware. Generally, the fast path of IP routing requires the following functions: IP packet validation, destination address parsing and table lookup, packet lifetime control (TTL update), and checksum calculation. The fast path may also be responsible for making packet classifications for QoS control and access filtering. Flows can be identified based on source IP address, destination IP address, TCP/UDP port numbers as well as IP Type of Service (TOS) field. Classification can even be based on higher layer packet attributes.

3.5.2 Non-critical Data Path Processing (Slow Path)

Packets destined to a router, such as maintenance, management or error protocol data are usually not time critical. However, they have to be integrated in an efficient way that does not interfere with the packet processing and, thus, does not slow down the time-critical path. Typical examples of these *non-time critical processing* tasks are error protocols (e.g., ICMP), routing protocols (e.g., RIP, OSPF, BGP), and network management protocols (e.g., SNMP). These processing tasks need to be centralized in a router node and typically reside above the network or transport protocols.

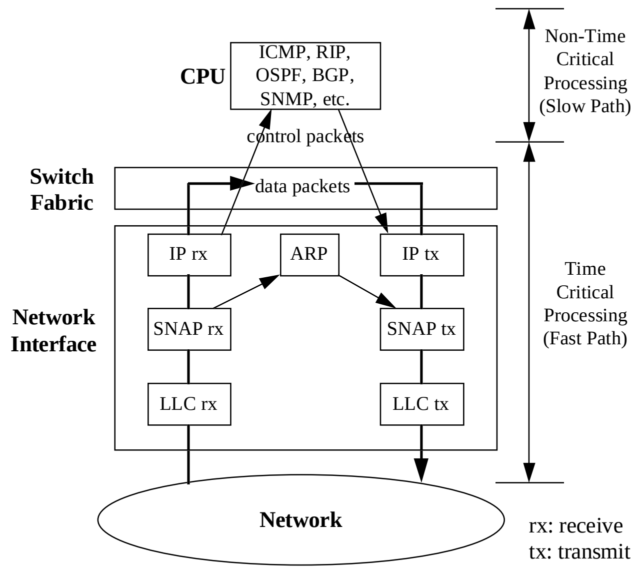


Figure 7. Example IEEE 802 protocol entities in an IP router [Adapted from 34].

As shown in Figure 7, only protocols in the forwarding path of a packet through the IP router are implemented on the network interface itself. Other protocols such as routing and network management protocols are implemented on the CPU. This way, the CPU does not adversely affect performance because it is located out of the data path, where it maintains route tables and determines the policies and resources used by the network interfaces. As an example, the CPU subsystem can be attached to the switch fabric in the same way as a regular network interface. In this case, the CPU subsystem is viewed by the switch fabric as a regular network interface. It has, however, a completely different internal architecture and function. This subsystem receives all non-critical protocol data units and requests to process certain related protocol entities (ICMP, SNMP, TCP, UDP, and the routing protocol entities RIP, OSPF, BGP, etc.). Any protocol data unit that needs to be sent on any network by these protocol entities is sent to the proper network interface, as if it was just another IP datagram relayed from another network.

The CPU subsystem can communicate with all other network interfaces through the exchange of coded messages across the switch fabric (or on a separate control bus [30][33]). IP datagrams generated by the CPU protocol entities are also coded in the same format. They carry the IP address of the next hop. For that, the CPU needs to access its individual routing table. This routing table can be the master table of the entire router. All other routing tables in the network interfaces will be exact replicas (or summaries in compressed table format) of the master table. Routing table updates in the network interfaces can then be done by broadcasting (if the switch fabric is capable of that) or any other suitable data push technique. Any update information (e.g., QoS policies, access control policies, packet drop policies, etc.) originated in the CPU has to be broadcast to all network interfaces. Such special data segments are received by the network interfaces which takes care of the actual write operation in their forwarding tables. Updates to the routing table in the CPU are done either by the various routing protocol entities or by management action. This centralization is reasonable since routing changes are assumed to happen infrequently and not particularly time critical. The CPU can also be configured to handle any packet whose destination address cannot be found in the forwarding table in the network interface card.

3.5.3 Fast Path or Slow Path?

It is not always obvious which router functions are to be implemented in the fast path or slow path. Some router designers may choose to include the ARP processing in the fast path instead of in the slow path of a router for performance reasons, and because ARP needs direct access to the physical network. Other may argue for ARP implementation in the slow path instead of the fast path. For example, in the ARP used for Ethernet, if a router gets a datagram to an IP address whose Ethernet address it does not know, it is supposed to send an ARP message and hold the datagram until it gets an ARP reply with the necessary Ethernet address. When the ARP is implemented in the slow path, datagrams for which the destination link layer address is unknown are passed to the CPU, which does the ARP and, once it gets the ARP reply, forwards the datagram and incorporates the link-layer address into future forwarding tables in the network interfaces.

There are other functions which router designers may argue to be not critical and are more appropriate to be implemented in the slow path. IP packet fragmentation and reassembly, source routing option, route recording option, timestamp option, and ICMP message generation are examples of such functions. It can be argued that packets requiring these functions are rare and can be handled in the slow path: a practical product does not need to be able to perform “wire-speed” routing when infrequently used options are present in the packet. Since such packets having such options comprise a small fraction of the total traffic, they can be handled as exception conditions. As a result, such packets can be handled by the CPU, i.e., the slow path. For IP packet headers with error, generally, the CPU can instruct the inbound network interface to discard the errored datagram. In some cases, the CPU will generate an ICMP message [34]. Alternatively, in the cache-based scheme [31], templates of some common ICMP messages such as the TimeExceeded message are kept in the forwarding engine and these can be combined with the IP header to generate a valid ICMP message.

An IP packet can be fragmented by a router, that is, a single packet can arrive, thereby resulting in multiple, smaller packets being transmitted onto the output ports. This capability allows a router to forward packets between ports where the output is incapable of carrying a packet of the desired length; that is, the MTU of the output port is less than that of the input port. Fragmentation is good in the sense that it allows communication between end systems connected through links with dissimilar MTUs. It is bad in that it imposes a significant processing burden on the router, which must perform more work to generate the resulting multiple output datagrams from the single input IP datagram. It is also bad from a data throughput point of view because, when one fragment is lost, the entire IP datagram must be retransmitted. The main arguments for implementing fragmentation in the slow path is that IP packet fragmentation can be considered an “exception condition”, outside of the fast path. Now that IP MTU discovery [11] is prevalent, fragmentation should be rare.

Reassembly of fragments may be necessary for packets destined for entities within the router itself. These fragments may have been generated either by other routers in the path between the sender and the router in question or by the original sending end system itself. Although fragment reassembly can be a resource-intensive process (both in CPU cycles and memory), the number of packets sent to the router is normally quite low relative to the number of packets being routed through. The number of fragmented packets destined for the router is a small percentage of the total router traffic. Thus, the performance of the router for packet reassembly is not critical and can be implemented in the slow path.

Figure 8 further categorizes the slow path router functions into two: those performed on a packet-by-packet basis (that is, optional or exception conditions) and those performed as background tasks.

Typical Router Slow Path Functions	
Packet-by-Packet Operations	**Background Tasks**
- Fragmentation and reassembly - Source routing option - Route recording option - Timestamp option - ICMP message generation	- Routing protocols (RIP, OSPF, BGP, etc.) - Network management (SNMP) - Router configuration (BOOTP, DHCP, etc.)

Figure 8. IP router slow-path functions.

3.5.4 Protocol Entities and IP Processing in the Distributed Router Architecture

The IP protocol is the most extensive entity in the packet processing path of an IP router and, thus, IP processing typically determines the achievable performance of a router. Therefore, a decomposition of IP that enables efficient multiprocessing is needed in the distributed router architecture. An example of a typical functional partitioning in the distributed router architecture is shown in Figure 9.

This distributed multiprocessing architecture, means that the various processing elements can work in parallel on their own tasks with little dependence on the other processors in the system. This architecture decouples the tasks associated with determining routes through the network from the time-critical tasks associated with IP processing. The results of this is an architecture with high levels of aggregate system performance and the ability to scale to increasingly higher performance levels.

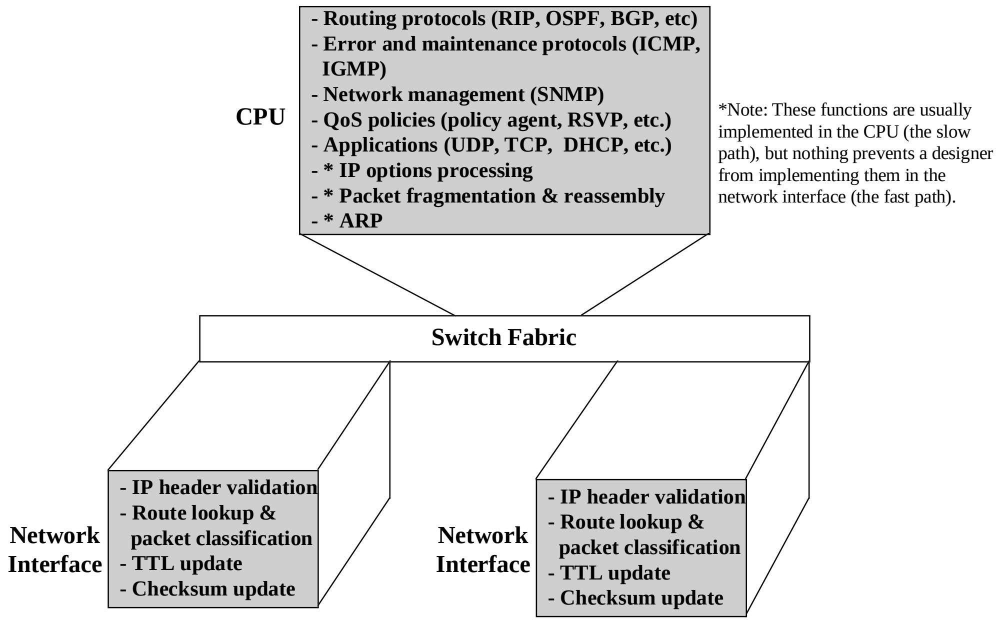

CPU
- Routing protocols (RIP, OSPF, BGP, etc)
- Error and maintenance protocols (ICMP, IGMP)
- Network management (SNMP)
- QoS policies (policy agent, RSVP, etc.)
- Applications (UDP, TCP, DHCP, etc.)
- * IP options processing
- * Packet fragmentation & reassembly
- * ARP
*Note: These functions are usually implemented in the CPU (the slow path), but nothing prevents a designer from implementing them in the network interface (the fast path).
Switch Fabric
Network Interface
- IP header validation
- Route lookup & packet classification
- TTL update
- Checksum update
Network Interface
- IP header validation
- Route lookup & packet classification
- TTL update
- Checksum update

Figure 9. An example functional partitioning in the distributed router architecture.

A high level diagram of a distributed router architecture is shown in Figure 10. Network interface cards built with general-purpose processors and complex communication protocols tend to be more expensive than those built using ASICs and simple communication protocols. Choosing between ASICs and general-purpose processors for an interface card is not straightforward. General-purpose processors tend to be more expensive, but allow extensive port functionality. They are also available off-the-shelf, and their price/performance ratio improves yearly [47]. 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 route processor (e.g., ARP, fragmentation and reassembly, IP options, etc.).

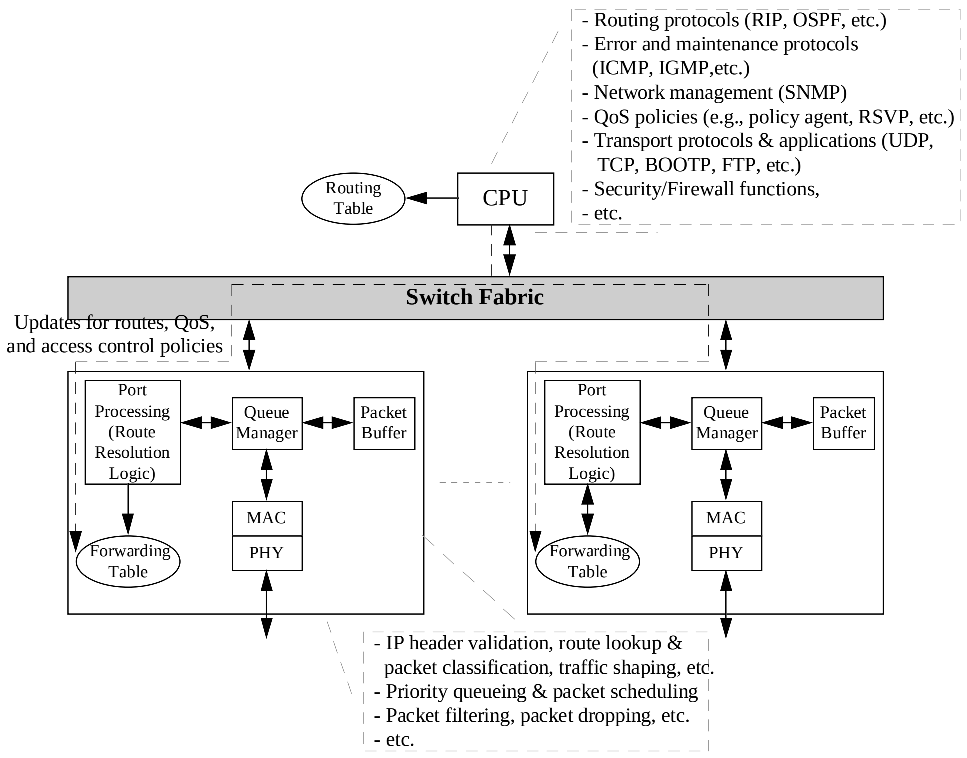

- Routing protocols (RIP, OSPF, etc.)
- Error and maintenance protocols (ICMP, IGMP,etc.)
- Network management (SNMP)
- QoS policies (e.g., policy agent, RSVP, etc.)
- Transport protocols & applications (UDP, TCP, BOOTP, FTP, etc.)
- Security/Firewall functions,
- etc.
Routing Table
CPU
Switch Fabric
Updates for routes, QoS, and access control policies
Port Processing (Route Resolution Logic)
Queue Manager
Packet Buffer
MAC
PHY
Forwarding Table
Port Processing (Route Resolution Logic)
Queue Manager
Packet Buffer
MAC
PHY
Forwarding Table
- IP header validation, route lookup & packet classification, traffic shaping, etc.
- Priority queueing & packet scheduling
- Packet filtering, packet dropping, etc.
- etc.

Figure 10. A high level functional diagram of a distributed router architecture.

Some router designers often observe that the IPv4 specification is very stable and say that it would be more cost effective to implement the forwarding engine in an ASIC. It is argued that ASIC can reduce the complexity on each system board by combining a number of functions into individual chips that are designed to perform at high speeds. Other designers also observe that the Internet is constantly evolving in a subtle way that require programmability and as such a fast processor is appropriate for the forwarding engine.

The forwarding database in a network interface consists of several cross-linked tables as illustrated in Figure 11. This database can include IP routes (unicast and multicast), ARP tables, and packet filtering information for QoS and security/access control.

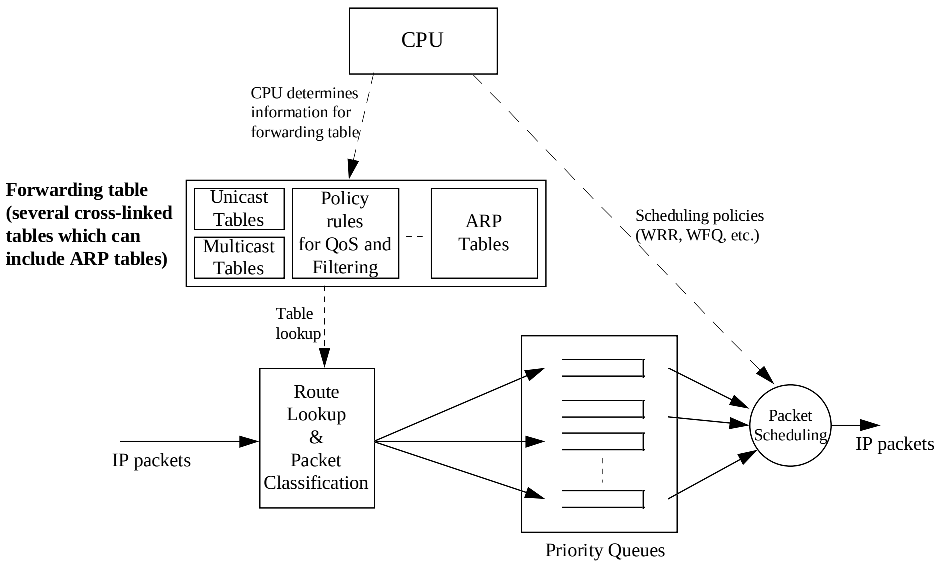

CPU
CPU determines information for forwarding table
Forwarding table (several cross-linked tables which can include ARP tables)
Unicast Tables
Multicast Tables
Policy rules for QoS and Filtering
ARP Tables
Scheduling policies (WRR, WFQ, etc.)
Table lookup
Route Lookup & Packet Classification
IP packets
Packet Scheduling
IP packets
Priority Queues

Figure 11. Forwarding database consisting of several cross-linked tables.

Now, let us take a generic shared memory router architecture and then trace the path of an IP packet as it goes through an ingress port and out of an egress port. The IP packet processing steps are shown in Figure 12.

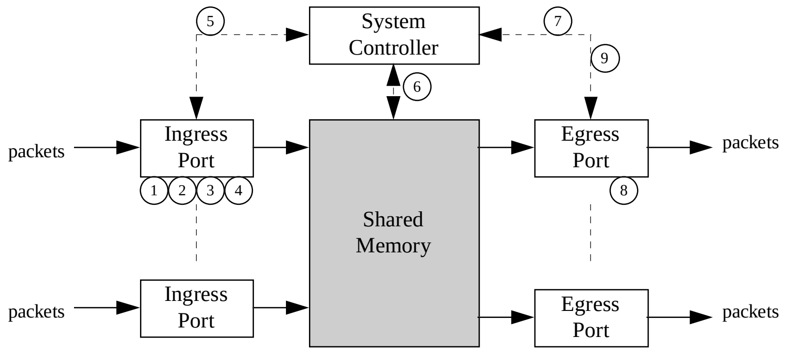


Figure 12. IP packet processing in a shared memory router architecture.

The IP packet processing steps are as follows:

1. *IP Header Validation*: As a packet enters an ingress port, the forwarding logic verifies all Layer 3 information (header length, packet length, protocol version, checksum, etc.).

2. *Route Lookup and Header Processing*: The router then performs an IP address lookup using the packet's destination address to determine the egress (or outbound) port, and performs all IP forwarding operations (TTL decrement, header checksum, etc.).

3. *Packet Classification*: In addition to examining the Layer 3 information, the forwarding engine examines Layer 4 and higher layer packet attributes relative to any QoS and access control policies.

4. With the Layer 3 and higher layer attributes in hand, the forwarding engine performs one or more parallel functions:

    - associates the packet with the appropriate priority and the appropriate egress port(s) (an internal routing tag provides the switch fabric with the appropriate egress port information, the QoS priority queue the packet is to be stored in, and the drop priority for congestion control),

    - redirects the packet to a different (overridden) destination (ICMP redirect),

    - drops the packet according to a congestion control policy (e.g., RED, WRED, etc.), or a security policy, and

    - performs the appropriate accounting functions (statistics collection, etc.).

5. The forwarding engine notifies the system controller that a packet has arrived.

6. The system controller reserves a memory location for the arriving packet.

7. Once the packet has been passed to the shared memory, the system controller signals the appropriate egress port(s). For multicast traffic, multiple egress ports are signalled.

8. The egress port(s) extracts the packet from the known shared memory location using any of a number of algorithms: Weighted Fair Queueing (WFQ), Weighted Round-Robin (WRR), Strict Priority (SP), etc.

9. When the destination egress port(s) has retrieved the packet, it notifies the system controller, and the memory location is made available for new traffic.

4. Typical Switch Fabrics of Routers

Switch fabric design is a very well studied area, especially in the context of ATM switches [48][49] so in this section, we examine briefly the most common fabrics used in router design. The switch fabric in a router is responsible for transferring packets between the other functional blocks. In particular, it routes user packets from the input modules to the appropriate output modules. The design of the switch fabric is complicated by other requirements such as multicasting, fault tolerance, and loss and delay priorities. When these requirements are considered, it becomes apparent that the switch fabric should have additional functions, e.g., concentration, packet duplication for multicasting if required, packet scheduling, packet discarding, and congestion monitoring and control.

Virtually all IP router designs are based on variations or combinations of the following basic approaches: shared memory; shared medium; distributed output buffered; space division (e.g., crossbar). Some important considerations for the switch fabric design are: throughput, packet loss, packet delays, amount of buffering, and complexity of implementation. For given input traffic, the switch fabric designs aim to maximize throughput and minimize packet delays and losses. In addition, the total amount of buffering should be minimal (to sustain the desired throughput without incurring excessive delays) and implementation should be simple.

4.1 Shared Medium Switch Fabric

In a router, packets may be routed by means of a shared medium e.g., bus, ring, or dual bus. The simplest switch fabric is the bus. Bus-based routers implement a monolithic backplane comprising a single medium over which all inter-module traffic must flow. Data is transmitted across the bus using Time Division Multiplexing (TDM), in which each module is allocated a time slot in a continuously repeating transmission. However, a bus is limited in capacity and by the arbitration overhead for sharing this critical resource. In a typical shared memory bus architecture, all ports access a central memory pool via a shared bus. An arbitration mechanism is used to control port access to the shared memory. The challenge is that it is almost impossible to build a bus arbitration scheme fast enough to provide nonblocking performance at multigigabit speeds.

Another example of a fabric using a time-division multiplexed (TDM) bus is shown in Figure 13. Incoming packets are sequentially broadcast on the bus (in a round-robin fashion). At each output, address filters examine the internal routing tag on each packet to determine if the packet is destined for that output. The address filters passes the appropriate packets through to the output buffers.

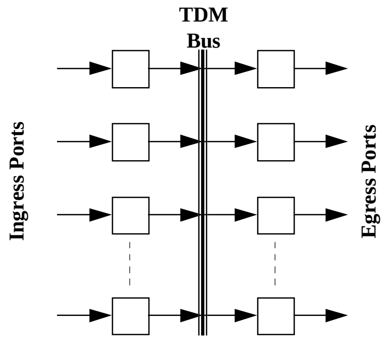


Figure 13. Shared medium bus: a TDM bus.

It is apparent that the bus must be capable of handling the total throughput. For discussion, we assume a router with *N* input ports and *N* output ports, with all port speeds equal to *S* (fixed size) packets per second. In this case, a packet time is defined as the time required to receive or transmit an entire packet at the port speed, i.e., 1/*S* sec. If the bus operates at a sufficiently high speed, at least *NS* packets/sec, then there are no conflicts for bandwidth and all queueing occurs at the outputs. Naturally, if the bus speed is less than *NS* packets/sec, some input queueing will probably be necessary.

The outputs are modular from each other, which has advantages in implementation and reliability. The address filters and output buffers are straightforward to implement. Also, the broadcast-and-select nature of this approach makes multicasting and broadcasting natural. For these reasons, the bus type switch fabric has found a lot of implementation in routers. However, the address filters and output buffers must operate at the speed of the shared medium, which could be up to *N* times faster than the port speed. There is a physical limit to the speed of the bus, address filters, and output buffers; these limit the scalability of this approach to large sizes and high speeds. Either the size *N* or speed S can be large, but there is a physical limitation on the product *NS*. As with the shared memory approach (to be discussed next), this approach involves output queueing, which is capable of the optimal throughput (compared to simple FIFO input queueing). However, the output buffers are not shared, and hence this approach requires more total amount of buffers than the shared memory fabric for the same packet loss rate.

4.2 Shared Memory Switch Fabric

A typical architecture of a shared memory fabric is shown in Figure 14. Incoming packets are typically converted from a serial to parallel form which are then written sequentially into a (dual port) random access memory. Their packet headers with internal routing tags are typically delivered to a memory controller, which decides the order in which packets are read out of the memory. The outgoing packets are demultiplexed to the outputs, where they are converted from parallel to serial form. Functionally, this is an output queueing approach, where the output buffers all physically belong to a common buffer pool. The output buffered approach is attractive because it can achieve a normalized throughput of one under a full load [50]. Sharing a common buffer pool has the advantage of minimizing the amounts of buffers required to achieve a specified packet loss rate. The main idea is that a central buffer is most capable of taking advantage of statistical sharing. If the rate of traffic to one output port is high, it can draw upon more buffer space until the common buffer pool is (partially or) completely filled.

Because the buffer space can be shared, this approach requires the minimum possible amount of buffering and has the most flexibility to accommodate traffic dynamics, in the sense that the shared memory can absorb large bursts directed to any output. For these reasons it is a popular approach for router design (e.g., [37][40][42][43][45]).

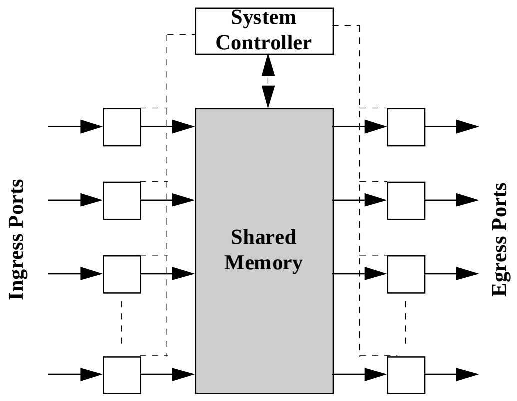


Figure 14. A shared memory switch fabric.

Unfortunately, the approach has its disadvantages. As the packets must be written into and read out from the memory one at a time, the shared memory must operate at the total throughput rate. It must be capable of reading and writing a packet (assuming fixed size packets) in every 1/*NS* sec, that is, *N* times faster than the port speed. As the access time of random access memories is physically limited, this speed-up factor *N* limits the ability of this approach to scale up to large sizes and fast speeds. Either the size *N* or speed *S* can be large, but the memory access time imposes a limit on the product *NS*, which is the total throughput. Moreover, the (centralized) memory controller must process (the routing tags of) packets at the same rate as the memory. This might be difficult if, for instance, the controller must handle multiple priority classes and complicated packet scheduling. Multicasting and broadcasting in this approach will also increase the complexity of the controller.

In shared memory switches, a single point of failure is invariably introduced in the design because adding a redundant switch fabric to this design is so complex and expensive. As a result, shared memory switch fabrics are best suited for small capacity systems.

4.3 Distributed Output Buffered Switch Fabric

The distributed output buffered approach is shown in Figure 15. Independent paths exist between all N^2 possible pairs of inputs and outputs. In this design, arriving packets are broadcast on separate buses to all outputs. Address filters at each output determine if the packets are destined for that output. Appropriate packets are passed through the address filters to the output queues.

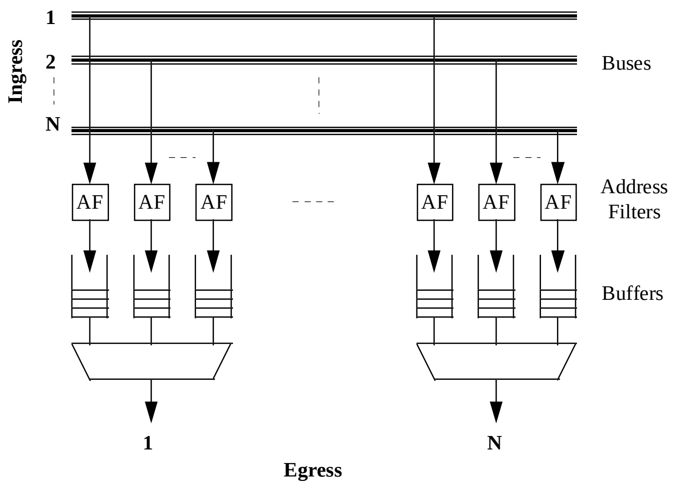


Figure 15. A distributed output buffered switch fabric.

This approach offers many attractive features. Naturally there is no conflict among the N^2 independent paths between inputs and outputs, and hence all queueing occurs at the outputs. As stated earlier, output queueing achieves the optimal normalized throughput compared to simple FIFO input queueing [50]. Like the shared medium approach, it is also broadcast-and-select in nature and, therefore, multicasting is natural. The address filters and output buffers are simple to implement. Unlike the shared medium approach, the address filters and buffers need to operate only at the port speed. All of the hardware can operate at the same speed. There is no speed-up factor to limit scalability in this approach. For these reasons, this approach has been taken in some commercial router designs (e.g., [41]).

Unfortunately, the quadratic N^2 growth of buffers means that the size N must be limited for practical reasons. However, in principle, there is no severe limitation on S. The port speed S can be increased to the physical limits of the address filters and output buffers. Hence, this approach might realize a high total throughput NS packets per second by scaling up the port speed S. The Knockout switch was an early prototype that suggested a trade-off to reduce the amount of buffers at the cost of higher packet loss [51]. Instead of

N buffers at each output, it was proposed to use only a fixed number L buffers at each output (for a total of NL buffers which is linear in N), based on the observation that the simultaneous arrival of more than L packets (cells) to any output was improbable. It was argued that $L = 8$ is sufficient under uniform random traffic conditions to achieve a cell loss rate of 10^{-6} for large N.

4.4 Space Division Switch Fabric: The Crossbar Switch

Optimal throughput and delay performance is obtained using output buffered switches. As long as input port and output port is under-subscribed, 100% throughput is achieved. Moreover, since upon arrival, the packets are immediately placed in the output buffers, it is possible to better control the latency of the packet. This helps in providing QoS guarantees. While this architecture appears to be especially convenient for providing QoS guarantees, it has serious limitations: the output buffered switch memory speed must be equal to at least the aggregate input speed across the switch. To achieve this, the switch fabric must operate at a rate at least equal to the aggregate of all the input links connected to the switch. However, increasing line rate (S) and increasing switch size (N) make it extremely difficult to significantly speedup the switch fabric, and also build memories with a bandwidth of order $O(NS)$.

At multigigabit and terabit speeds it becomes difficult to build output buffered switches. As a result some high-speed implementations are based on the input buffered switch architecture. One of the most popular interconnection networks used for building input buffered switches is the crossbar because of its (i) low cost, (ii) good scalability and (iii) non-blocking properties. Crossbar switches have an architecture that, depending on the implementation, can scale to very high bandwidths. Considerations of cost and complexity are the primary constraints on the capacity of switches of this type. The crossbar switch (see Figure 16) is a simple example of a space division fabric which can physically connect any of the N inputs to any of the N outputs. An input buffered crossbar switch has the crossbar fabric running at the link rate. In this architecture buffering occurs at the inputs, and the speed of the memory does not need to exceed the speed of a single port. Given the current state of technology, this architecture is widely considered to be substantially more scalable than output buffered or shared memory switches. This has renewed interest in switches with lower complexity (and cost) such as input buffered switches despite their deficiencies. However, the crossbar architecture presents many technical challenges that need to be overcome in order to provide bandwidth and delay guarantees. Examples of commercial routers that use crossbar switch fabrics are [38][39][45].

We start with the issue of providing bandwidth guarantees in the crossbar architecture. For the case when there is a single FIFO queue at each input, it has long been known that a serious problem referred to as head-of-line (HOL) blocking [50] can substantially reduce achievable throughput. In particular, the well-known results of [50] is that for uniform random distribution of input traffic, the achievable throughput is only 58.6%. Moreover, Li [52] has shown that the maximum throughput of the switch decreases monotonically with increasing burst size. Considerable amount of work has been done in recent years to build input buffered switches that match the performance of an output buffered switch. One way of reducing the effect of HOL blocking is to increase the speed of the input/output channel (i.e., the speedup of the switch fabric). Speedup is defined as the ratio of the switch fabric bandwidth and the bandwidth of the input links. There have been a number of studies such as [53][54] which showed that an input buffered crossbar switch with a single FIFO at the input can achieve about 99% throughput under certain assumptions on the input traffic statistics for speedup in the range of 4 - 5. A more recent simulation study [55] suggested that speedup as low as 2 may be sufficient to obtain performance comparable to that of output buffered switches.

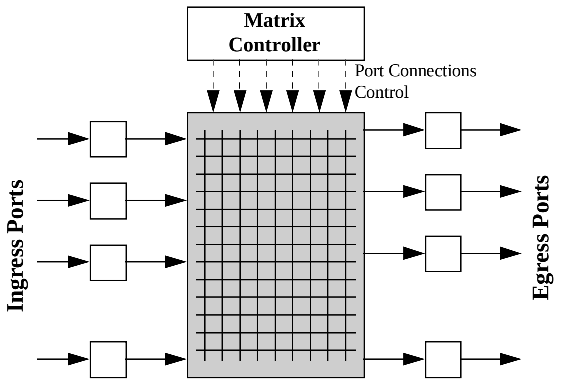


Figure 16. A crossbar switch.

Another way of eliminating the HOL blocking is by changing the queueing structure at the input. Instead of maintaining a single FIFO at the input, a separate queue per each output can be maintained at each input. To eliminate HOL blocking, virtual output queues (VOQs) were proposed at the inputs. However, since there could be contention at the inputs and outputs, there is a necessity for an arbitration algorithm to schedule packets between various inputs and outputs (equivalent to the matching problem for bipartite graphs). It has been shown that an input buffered switch with VOQs can provide asymptotic 100% throughput using a maximum matching algorithm [56]. However, the complexity of the best known maximum match algorithm is too high for high speed implementation. Moreover, under certain traffic conditions, maximum matching can lead to starvation. Over the years, a number of maximal matching algorithms have been proposed [57][58][59][60][61].

As stated above, increasing the speedup of the switch fabric can improve the performance of an input buffered switch. However, when the switch fabric has a higher bandwidth than the links, buffering is required at the outputs too. Thus a combination of input buffered and output buffered switch is required, i.e., CIOB (Combined Input and Output Buffered). The goal of most designs then is to find the minimum speedup required to match the performance of an output buffered switch using a CIOB and VOQs. McKeown *et al.* [62] shown that a CIOB switch with VOQs is always work conserving if speedup is greater $N/2$. In a recent work, Prabhakar *et al.* [63] showed that a speed of 4 is sufficient to emulate an output buffered switch (with an output FIFO) using a CIOB switch with VOQs.

4.5 Other Issues in Router Switch Fabric Design

We have described above four typical design approaches for router switch fabrics. Needless to say, endless variations of these designs can be imagined but the above are the most common fabrics found in routers. There are other issues applicable to understanding the trade-offs involved in any new design. We discuss some of these issues next.

4.5.1 Construction of Large Router Switch Fabrics

With regards to the construction of large switch fabrics, most of the four basic switch fabric design approaches are capable of realizing routers of limited throughput. The shared memory and shared medium approaches can achieve a throughput limited by memory access time. The space division approach has no special constraints on throughput or size, only physical factors do limit the maximum size in practice. There are physical limits to the circuit density and number of input/output (I/O) pins. Interconnection complexity and power dissipation become more difficult issues with fabric size. In addition, reliability and repairability become difficult with size. Modifications to maximize the throughput of space division fabrics to address HOL blocking increases the implementation complexity.

It is generally accepted that large router switch fabrics of 1 terabits per second (Tbps) throughput or more cannot be realized simply by scaling up a fabric design in size and speed. Instead, large fabrics must be constructed by interconnection of switch modules of limited throughput. The small modules may be designed following any approach, and there are various ways to interconnect them.

4.5.2 Fault Tolerance and Reliability

With the rapid growth of the Internet and the emergence of growing competition between Internet Service Providers (ISPs), reliability has become an important issue for IP routers. In addition, multigigabit routers will be deployed in the core of enterprise networks and the Internet. Traffic from thousands of individual flows pass through the switch fabric at any given time [32]. Thus, the robustness and overall availability of the switch fabric becomes a critically important design parameter. As in any communication system, fault tolerance is achieved by adding redundancy to the crucial components. In a router, one of the most crucial components is the packet routing and buffering fabric. In addition to redundancy, other considerations include detection of faults and isolation and recovery.

4.5.3 Multicasting

New applications or services are emerging that utilize multicast transport. These applications include distribution of news, financial data, software, video, audio and multi-person conferencing. These services or applications will require a router to multicast an incoming packet to a number of selected outputs or broadcast it to all outputs. Multicasting is inherently natural to the shared medium and distributed output-buffered approaches. Both approaches consist of broadcasting incoming packets and selecting the appropriate packets with address filters at the output buffers. For multicasting, an address filter can recognize a set of multicast addresses as well as output port addresses. As a result, multicasting is natural in these two broadcast-and-select approaches.

Multicasting is not natural to the shared memory approach but can be implemented with additional control circuitry. A multicast packet may be duplicated before the memory or read multiple times from the memory. The first approach obviously requires more memory because multiple copies of the same packet are maintained in the memory. In the second approach, a packet is read multiple times from the same memory location. The control circuitry must keep the packet in memory until it has been read to all the output ports in the multicast group.

Multicast in the space division fabrics is simple to implement but has some consequences. For example, a crossbar switch (with input buffering) is naturally capable of broadcasting one incoming packet to multiple outputs. However, this would aggravate the HOL blocking at the input buffers. Approaches to alleviate the HOL blocking effect would increase the complexity of buffer control. Other inefficient approaches in crossbar switches require an input port to write out multiple copies of the packet to different output ports one at a time. This does not support the one-to-many transfers required for multicasting as in the shared bus architecture and the fully distributed output buffered architectures. The usual concern about making multiple copies is that it reduces effective switch throughput. Several approaches for handling multicasting in crossbar switches have been proposed [64]. Generally, multicasting increases the complexity of space division fabrics.

4.5.4 Buffer Management and Quality of Service (QoS)

The prioritization of mission critical applications and the support of IP telephony and video conferencing create the requirement for supporting QoS enforcement with the switch fabric. These applications are sensitive to both absolute latency and latency variations.

Beyond best-effort service, routers are beginning to offer a number of QoS or priority classes. Priorities are used to indicate the preferential treatment of one traffic class over another. The switch fabrics must handle these classes of traffic differently according to their QoS requirements. In the output buffered switch fabric, for example, typically the fabric will have multiple buffers at each output port and one buffer for each QoS traffic class. The buffers may be physically separate or a physical buffer may be divided logically into separate buffers.

Buffer management here refers to the discarding policy for the input of packets into the buffers (e.g., Drop Tail, Drop-From-Front, Random Early Detection (RED), etc.), and the scheduling policy for the output of packets from the buffers (e.g., strict priority, weighted round-robin (WRR), weighted fair queueing (WFQ), etc.). Buffer management in the IP router involves both dimensions of time (packet scheduling) and buffer space (packet discarding). The IP traffic classes are distinguished in the time and space dimensions by their packet delay and packet loss priorities. We therefore see that buffer management and QoS support is an integral part of the switch fabric design.

5. Conclusions and Open Problems

IP provides an amazing degree of flexibility in building large and arbitrary complex networks. Interworking routers capable of forwarding aggregate data rates in the multigigabit and terabit per second range are required in emerging high performance networking environments. This paper has presented an evaluation of typical approaches proposed for designing high speed routers. We have focused primarily on the architectural overview and the design of the components that have the highest effect on performance.

First, we have observed that high-speed routers need to have enough internal bandwidth to move packets between its interfaces at multigigabit and terabit rates. The router design should use a switched backplane. Until very recently, the standard router used a shared bus rather than a switched backplane. While bus-based routers may have satisfied the early needs of IP networks, emerging demands for high bandwidth, QoS delivery, multicast, and high availability place the bus architecture at a significant disadvantage. For high speeds, one really needs the parallelism of a switch with superior QoS, multicast, scalability, and robustness properties. Second, routers need enough packet processing power to forward several million packets per second (Mpps). Routing table lookups and data movements are the major consumers of processing cycles. The processing time of these tasks does not decrease linearly if faster processors are used. This is because of the sometimes dominating effect of memory access rate.

Experience has shown that while an IP router must, in general, perform a myriad of functions, in practice the vast majority of packets need only a few operations performed in real-time. Thus, the performance critical functions can be implemented in hardware (the fast path) and the remaining (necessary, but less time-critical) functions in software (the slow path). IP contains many features and functions that are either rarely used or that can be performed in the background of high-speed data forwarding (for example, routing protocol operation and network management). The router architecture should be optimized for those functions that must be performed in real-time, on a packet-by-packet basis, for the majority of the packets. This creates an optimized routing solution that route packets at high speed at a reasonable cost.

It has been observed in [47] that the cost of a router port depends on, 1) the amount and kind of memory it uses, 2) its processing power, and 3) the complexity of the protocol used for communication between the port and the route processor. This means the design of a router involve trade-offs between performance, complexity, and cost.

Router ports built with general-purpose processors and complex communication protocols tend to be more expensive than those built using ASICs and simple communication protocols. Choosing between ASICs and general-purpose processors for an interface card is not straightforward. General-purpose processors tend to be more expensive, but allow extensive port functionality. They are also available off-the-shelf, and their price/performance ratio improves yearly. 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 route processor.

The cost of a router port is also proportional to the type and size of memory on the port. SRAMs offer faster access times, but are more expensive than 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 to increase their transmission windows without excessive losses.

The cost of a router port is also determined by the complexity of the internal connections between the control paths and the data paths in the port card. In some designs, a centralized controller sends commands to each port through the switch fabric and the port's internal buffers. Careful engineering of the control protocol is necessary to reduce the cost of the port control circuitry and also the loss of command packets which will certainly need retransmission.

Significant advances have been made in router designs to address the most demanding customer issues regarding high speed packet forwarding (e.g., route lookup algorithms, high-speed switching cores and forwarding engines), low per-port cost, flexibility and programmability, reliability, and ease of configuration. While these advances have been made in the design of IP routers, some important open issues still remain to be resolved. These include packet classification and resource provisioning, improved price/performance router designs, “active networking” [65] and ease of configuration, reliability and fault tolerance designs, and Internet billing/pricing. Extensive work is being carried out both in the research community and industry to address these problems.

References

[1]. P. Newman, T. Lyon, and G. Minshall, “Flow Labelled IP: A Connectionless Approach to ATM,” *Proc IEEE Infocom’96*, San Francisco, CA, March 1996, pp. 1251 - 1260.

[2]. Y. Katsube, K. Nagami, and H. Esaki, “Toshiba’s Router Architecture Extensions for ATM: Overview,” *IETF RFC 2098*, April 1997.

[3]. Y. Rekhter, B. Davie, D. Katz, E. Rosen, and G. Swallow, “Cisco Systems’ Tag Switching Architecture Overview,” *IETF RFC 2105*, Feb. 1997.

[4]. F. Baker, “Requirements for IP Version 4 Routers,” *IETF RFC* 1812, Jun. 1995.

[5]. W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Reading, MA: Addison-Wesley, 1994.

[6]. C. Huitema, *Routing in the Internet*, Prentice Hall, 1996.

[7]. J. Moy, *OSPF: Anatomy of an Internet Routing Protocol*, 1998.

[8]. R. Braden, D. Borman, and C. Partridge, “Computing the Internet Checksum,” *IETF RFC 1071*, Sept. 1988.

[9]. T. Mallory and A. Kullberg, “Incremental Updating of the Internet Checksum,” *IETF RFC 1141*, Jan. 1990.

[10]. C. A. Kent and J. C. Mogul, “Fragmentation Considered Harmful,” *Computer Commun. Rev.*, Vol. 17, No. 5, Aug. 1987, pp. 390 - 401.

[11]. J. Mogul and S. Deering, “Path MTU Discovery” *IETF RFC 1191*, April 1990.

[12]. V. Fuller et al. “Classless Inter-Domain Routing,” *IETF RFC 1519*, Jun. 1993.

[13]. K. Sklower, “A Tree-Based Packet Routing Table for Berkeley Unix,” *USENIX, Winter’91*, Dallas, TX, 1991.

[14]. W. Doeringer, G. Karjoth, and M. Nassehi, “Routing on Longest-Matching Prefixes,” *IEEE/ACM Trans. on Networking*, Vol. 4, No. 1, Feb. 1996, pp. 86 - 97.

[15]. D. C. Feldmeier, “Improving Gateway Performance with a Routing-Table Cache,” *Proc. IEEE Infocom’88*, New Orleans, LI, Mar. 1988.

[16]. C. Partridge, “Locality and Route Caches,” *NSF Workshop on Internet Statistics Measurement and Analysis*, San Diego, CA, Feb. 1996.

[17]. D. Knuth, *The Art of Computer Programming, Vol. 3. Sorting and Searching*, Addison-Wesley, 1973.

[18]. M. Degermark, *et al*., “Small Forwarding Tables for Fast Routing Lookups,” *Proc. ACM SIGCOMM’97*, Cannes, France, Sept. 1997.

[19]. H.-Y. Tzeng, “Longest Prefix Search Using Compressed Trees,” *Proc. Globecom’98*, Sydney, Australia, Nov. 1998.

[20]. M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, “Scalable High Speed IP Routing Lookup,” *Proc. ACM SIGCOMM’97*, Cannes, France, Sept. 1997.

[21]. V. Srinivasan and G. Varghese, “Faster IP Lookups using Controlled Prefix Expansion,” *Proc. ACM SIGMETRICS*, May 1998.

[22]. S. Nilsson and G. Karlsson, “Fast Address Look-Up for Internet Routers,” *Proc. of IEEE Broadband Communications’98*, April 1998.

[23]. E. Filippi, V. Innocenti, and V. Vercellone, “Address Lookup Solutions for Gigabit Switch/Router,” *Proc. Globecom’98*, Sydney, Australia, Nov. 1998.

[24]. A. J. McAuley and P. Francis, “Fast Routing Table Lookup using CAMs,” *Proc. IEEE Infocom’93*, San Francisco, CA, Mar. 1993, pp. 1382 - 1391.

[25]. T. B. Pei and C. Zukowski, “Putting Routing Tables in Silicon,” *IEEE Network*, Vol. 6, Jan. 1992, pp. 42 - 50.

[26]. M. Zitterbart et al., “HeaRT: High Performance Routing Table Lookup,” *4th IEEE Workshop on Architecture & Implementation of High Performance Communications Subsystems*, Thessaloniki, Greece, Jun. 1997.

[27]. P. Gupta, S. Lin, and N. McKeown, “Routing Lookups in Hardware at Memory Access Speeds,” *Proc. IEEE Infocom’98*, Mar. 1998.

[28]. S. F. Bryant and D. L. A. Brash, “The DECNIS 500/600 Multiprotocol Bridge/Router and Gateway,” *Digital Technical Journal*, Vol. 5, No. 1, 1993.

[29]. P. Marimuthu, I. Viniotis, and T. L. Sheu, “A Parallel Router Architecture for High Speed LAN Internetworking,” *17th IEEE Conf. on Local Computer Networks*, Minneapolis, Minnesota, Sept. 1992.

[30]. S. Asthana, C. Delph, H. V. Jagadish, and P. Krzyzanowski, “Towards a Gigabit IP Router,” *Journal of High Speed Networks*, Vol. 1, No. 4, 1992.

[31]. C. Partridge et al., “A 50Gb/s IP Router,” *IEEE/ACM Trans. on Networking*, Vol. 6, No. 3, Jun 1998, pp. 237 - 248.

[32]. K. Thomson, G. J. Miller, and R. Wilder, “Wide-Area Traffic Patterns and Characteristics,” *IEEE Network*, Dec. 1997.

[33]. V. P. Kumar, T. V. Lakshman, and D. Stiliadis, “Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow’s Internet,” *IEEE Commun. Mag.*, May 1998, pp. 152 - 164.

[34]. A Tantawy, O Koufopavlou, M. Zitterbart, and J. Abler, “On the Design of a Multigigabit IP Router,” *Journal of High Speed Networks*, Vol. 3, 1994, pp. 209 - 232.

[35]. O Koufopavlou, A. Tantawy, and M. Zitterbart, “IP-Routing among Gigabit Networks,” *Interoperability in Broadband Networks*, S. Rao (Ed.), IOS Press, 1994, pp. 282 - 289.

[36]. O Koufopavlou, A. Tantawy, and M. Zitterbart, “A Comparison of Gigabit Router Architectures,” *High Performance Networking*, E. Fdida (Ed.), Elsevier Science B. V. (North-Holland), 1994.

[37]. “Implementing the Routing Switch: How to Route at Switch Speeds and Switch Costs”, *White Paper*, Bay Networks, 1997.

[38]. “Cisco 12000 Gigabit Switch Router,” *White Paper*, Cisco Systems, 1997.

[39]. “Performance Optimized Ethernet Switching,” *Cajun White Paper #1*, Lucent Technologies.

[40]. “Internet Backbone Routers and Evolving Internet Design,” *White Paper*, Juniper Networks, Sept. 1998.

[41]. “The Integrated Network Services Switch Architecture and Technology,” *White Paper*, Berkeley Networks, 1997.

[42]. “Torrent IP9000 Gigabit Router,” *White Paper*, Torrent Networking Technologies, 1997.

[43]. “Wire-Speed IP Routing,” *White Paper*, Extreme Networks, 1997.

[44]. “PE-4884 Gigabit Routing Switch,” *White Paper*, Packet Engines, 1997.

[45]. “GRF 400 White Paper: A Practical IP Switch for Next-Generation Networks,” *White Paper*, Ascend Communications, 1998.

[46]. "Rule Your Networks: An Overview of StreamProcessor Applications," *White Paper*, NEO Networks, 1997.

[47]. S. Keshav and R. Sharma, "Issues and Trends in Router Design," *IEEE Commun. Mag.*, May 1998, pp. 144 - 151.

[48]. H. Ahmadi and W. Denzel, "A Survey of Modern High-Performance Switching Techniques," *IEEE J. on Selected Areas in Commun.*, Vol. 7, Sept. 1989, pp. 1091 - 1103.

[49]. F. Tobagi, "Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks," *Proc. of the IEEE*, Vol. 78, Jan. 1990, pp. 133 - 178.

[50]. M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queueing on a Space-Division Packet Switch," *IEEE Trans. on Commun.*, Vol. COM-35, Dec. 1987, pp. 1337 - 1356.

[51]. Y.-S. Yeh, M. Hluchyj and A. S. Acampora, "The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching," *IEEE J. on Selected Areas in Commun*. Vol. SAC-5, No. 8, Oct. 1987, pp. 1274 - 1282.

[52]. S.-Q. Li, "Performance of a Non-blocking Space-division Packet Switch with Correlated Input Traffic," *Proc. IEEE Globecom'89*, 1989, pp. 1754 - 1763.

[53]. C.-Y. Chang, A. J. Paulraj, and T. Kailath, "A Broadband Packet Switch Architecture with Input and Output Queueing," *Proc. Globecom'94*, 1994.

[54]. I. Iliadis, and W. Denzel, "Performance of Packet Switches with Input and Output Queueing," *Proc. ICC'90*, 1990.

[55]. R. Guerin and K. N. Sivarajan, "Delay and Throughput Performance of Speed-Up Input-Queueing Packet Switches," *IBM Research Report RC 20892*, Jun. 1997.

[56]. N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *Proc. IEEE Infocom'96*, 1996, pp. 296 - 302.

[57]. T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. on Computer Systems*, Vol. 11, No. 4, Nov. 1993, pp. 319 - 352.

[58]. D. Stiliadis and A. Verma, "Providing Bandwidth Guarantees in an Input-Buffered Crossbar Switch," *Proc. IEEE Infocom'95*, 1995, pp. 960 - 968.

[59]. N. McKeown, "Scheduling Algorithms for Input-Queued Cell Switches," *Ph.D. Thesis*, UC Berkeley, May 1995.

[60]. C. Lund, S. Phillips, and N. Reingold, "Fair Prioritized Scheduling in an Input-Buffered Switch," *Proc. Broadband Communications*, 1996.

[61]. A. Mekkittikul and N. McKeown, "A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches," *Proc. IEEE Infocom'98*, Mar. 1998.

[62]. N. McKeown, B. Prabhakar, and M. Zhu, "Matching Output Queueing with Combined Input and Output Queueing," *Proc. 35th Annual Allerton Conf. on Communications, Control and Computing*, Oct. 1997.

[63]. B. Prabhakar and N. McKeown, "On the Speedup Required for Combined Input and Output Queueing Switching," Computer Systems Lab, *Technical Report CSL-TR-97-738*, Stanford University.

[64]. N. McKeown, "Fast Switched Backplane for a Gigabit Switched Router," *Technical Report*, Dept. of Elect. Eng., Stanford University.

[65]. D. Tennenhouse, et al., "A Survey of Active Network Research," *IEEE Commun. Mag.*, Jan. 1997.