Anatomy of a Gigabit Router

By Peter Newman
June 1998

THE RECENT GROWTH and public acceptance of the Internet has provoked intense interest in the design of high performance, IP-only (or IP-mostly) routers, often called gigabit routers. Such routers employ hardware in the data path both for switching data packets and to speed up the packet header lookup process. This article defines some of the components of a gigabit router and investigates some of the design decisions involved.

Overview.

A gigabit router consists of three main components: line cards, switch fabric, and a controller. (In fact all communications switches, e.g. TDM circuit switch, ATM switch, Ethernet switch, etc., may be considered to consist of these same three components.)

Line Cards. The line cards contain the input and output ports. These are the interfaces to the electrical or optical transmission links that carry the data packets. A single line card may contain only one input and one output port or it may contain many. The number of ports depends on the physical dimensions of the connector and the bit rate of the transmission link. Typically all of the ports on a single line card are of the same type (e.g. 10/100 Ethernet, DS-3, T-1, OC-3c, etc.).

Switch Fabric. The switch fabric interconnects the line cards. For systems with an aggregate capacity of less than about 1 gigabits per second (Gbps) the switch fabric may be implemented with a backplane bus. For a switch fabric with an aggregate capacity of up to 20 Gbps a crossbar or a shared memory is currently popular. For an aggregate capacity in excess of 20 Gbps a crossbar or a multi-stage switch fabric is generally used.

Controller. The controller contains the computing power required to manage the router. It implements the control path of the router and does not participate in the data path, i.e. the forwarding of normal data packets. The controller runs the routing protocols — complex pieces of software that discover where things are in a network. They figure out in which direction packets should be forwarded to reach any requested destination in the network. The routing protocols do not forward the actual data packets themselves. The information discovered via the routing protocols is compiled into a forwarding table and the forwarding table is used to forward the data packets.

The complexity of a routing protocol grows with the number of ports on the router and not with the speed of these ports. So we can run the routing protocols in a single centralized processor even for routers of very high aggregate capacity. The packet forwarding function however is directly related to the number of packets per second that can be processed. To achieve high aggregate forwarding rates we can use parallelism. We can implement multiple packet forwarding engines each with an identical copy of the forwarding table supplied by the controller.

Centralized vs. Distributed.

This brings us to the first major design decision, centralized or distributed forwarding. In the centralized architecture we use hardware techniques to implement an extremely fast packet forwarding engine. All packet forwarding functions are implemented in this single centralized function.

Alternatively we can distribute the forwarding function into a number of lower speed packet forwarding engines. A popular distribution is to locate one forwarding engine on each line card, though it is possible to implement a central pool of forwarding engines shared between the line cards in some fashion.

It is a general rule in hardware that up to a point it is better to implement a function in a single high-speed device than in multiple lower-speed devices. It results in lower overall costs and is far easier to manage. There is however a threshold beyond which it becomes increasingly expensive to achieve higher speeds in a single centralized function and thus parallelism must be employed to achieve higher aggregate speed. We see this for example in the switch fabric. Up to about 10 or 20 Gbps a single shared memory is popular, but beyond this capacity switch fabrics that make use of parallelism are employed. The same applies to the packet forwarding function. Beyond something like 10 million packets per second (Mpps) it is difficult to implement the forwarding function in a single piece of hardware and multiple forwarding engines must be used.

If the forwarding engine is centralized the customer is forced to purchase the entire maximum forwarding capacity even if they only want to attach a few line cards. If the forwarding engine is distributed the customer can purchase forwarding capacity when they need it, i.e. when they purchase more line cards.

Distributed forwarding is optimized (in terms of cost) for the low end, when only a few line cards are purchased. Centralized forwarding is cost-optimized for the high end, when the system is almost fully stuffed with line cards. Exactly the same argument can be made for the switch fabric, although here it is much harder to distribute efficiently because it provides the central interconnection function.

Switching Capacity Metrics.

At this point we should mention the difference between the performance of a gigabit router in terms of its packet forwarding capability (Mpps) and its aggregate capacity (Gbps). The two are not the same. The aggregate capacity comes from the switch fabric, the ability to transmit bits across the system. People have been building packet switches with high aggregate capacity for about ten years so it is fairly well understood. The packet forwarding capability comes from the packet forwarding engine.

We have only recently begun investigating packet forwarding in hardware. So some of the gigabit router products available today have an aggregate capacity from the switch fabric of which the user will only ever be able to use a modest percentage because the packet forwarding engine cannot forward packets fast enough. Of course, if all of the packets are of maximum size the aggregate capacity can be fully used. However, 40 percent of the packets on the Internet today are of minimum size — TCP ACKs [1].

Naming of Parts [2]

Now that we have given a brief overview of a gigabit router we can look at some of the functions proposed for the data path. In this section we define them and give a little background. In the following section we discuss the acceleration of these functions by applying hardware.

Resource Reservation. Back when virtual circuits were cool, we had a simple method of associating the incoming traffic with the resources that were available to it. For example, one could reserve a specified bandwidth on an outgoing link and make that bandwidth available only to the traffic on a specified virtual circuit. Of course this approach had its drawbacks; it took a long time for the control software (signalling) to set up the virtual circuit and reserve the bandwidth across the network, and state (control information) had to be installed in every switching device in the path.

The Internet uses datagrams. But we would still like to be able to do some of the things that you could do with virtual circuits, like reserving resources. People have been trying to do this in a vaguely similar manner to that used in a virtual circuit network and have found it to be extremely difficult in practice, as the RSVP effort has demonstrated.

Recently, there has been a shift away from the IETF’s Integrated Services group (IntServ) approach of reserving resources to a simpler packet marking technique now being defined by the Differentiated Services (DiffServ) group. DiffServ is attempting to define something simple but useful that allows some packets to get faster service than others do.

Classification. Whether one is attempting to reserve precise resources or just trying to give some packets better service than others one needs a mechanism to decide which packets get what resources. This mechanism is called classification. In a virtual circuit network this is easy because the virtual circuit identifies the traffic. But in a datagram network one needs some other means of identifying the traffic. The only distinguishing marks on a packet are the fields in the packet header.

Finding Flows. The classification function looks at the fields in the packet header to group packets into classes. These classes are often referred to as flows. They are roughly the equivalent of a virtual circuit in a datagram network though much more flexible. Resources are allocated to these flows.

In IP version 6 a field in the header, the *Flow ID*, is defined to assist in the identification of flows. In IP version 4 the most useful fields are the source and destination addresses, the protocol field, and the type of service field (although the DiffServ group is currently redefining this field).

The source and destination port numbers are also useful from TCP or UDP to define flows of high granularity. The ability to mask out fields, and parts of fields, is required. (The ability to specify arbitrary ranges of values may also be useful, though this is much harder to achieve at high-speed in hardware.) For example, to group together all packets from a particular source network, the source address and netmask would be used. Entries in the classifier are called filters or rules and express the policy for handling particular classes of traffic.

Classification can be performed on the input interface, on the output interface, or on both input and output. It is possible for a single packet to match several filters in the classifier. In this case some policy must be established to determine what the result should be. Each filter in the classifier specifies how packets in that class should be handled. Packets matching a rule can be accepted or discarded. Special resources such as reserved bandwidth may be available to a class. Classes may be queued at different priorities, policed, traffic shaped, or queued in their own private queue in a per-flow queueing system. A class may even be routed differently compared to other packets to the same destination.

Longest Prefix Match.

Here again life was easy with a virtual circuit. You just performed a direct lookup of the virtual circuit identifier in a simple table and it told you to which output to send the traffic. But you can't directly look up a 32-bit destination IP address in single table. It would take 4 *billion* entries. And even if you could afford that much memory the management of the table would be unpleasant.

In the good old days when only nerds and scientists inhabited the Internet (up until the early '90s) this wasn't too hard. There were three lengths (classes) of the network component of the IP address: 8-bits, 16-bits, and 24-bits; and the address itself told you which class it belonged to. When the great unwashed were invited to join the Internet it was quickly realized that pretty soon we would run out of addresses if they continued to be allocated in the old manner, and also that the routing tables in the backbone routers were becoming too large.

IPV6. The solution was to begin work on IPV6, which has 128-bit addresses, and in the mean time to allow the network component of the IP version 4 address to be of any length. This so-called Classless InterDomain Routing (CIDR) approach permits IP addresses to be aggregated if the network is structured hierarchically. It's similar to the telephone network; e.g. all telephone numbers beginning +33 are in France.

The result of all this is that when a router receives a 32-bit destination IP address to look up in its forwarding table it does not know how many of the bits belong to the network component of the address.

In concept the router has to perform an address lookup up to 32 times, first with all 32 bits, then with the first 31 bits, then with the first 30 bits, and so on until it finds the longest match in its table. Of course it doesn't implement the lookup in this manner—typically a tree structure is employed. Much current effort is being applied to delevop algorithms that can implement this operation very fast.

Queueing.

In the Internet the transmission of packets from their sources is not scheduled. So packets tend to arrive at a switching node in bursts much like public transport (even if it is scheduled). Queues are required to store packets during the inevitable periods when the packet arrival rate exceeds the transmission rate of the output port. The queue drains when the arrival rate of packets is less than the transmission rate of the output port.

To allow TCP connections to open their window smoothly a rule of thumb is that a port should have sufficient buffer to queue one delay-bandwidth product's worth of data. However, if the queue is too large it just adds unnecessary delay if it fills. If the queue is too small packets are dropped when bursts arrive, resulting in poor behavior, low throughput for TCP and high loss for UDP.

Queueing consists of two operations: Putting the packets into the queue (enqueueing) and taking the packets out of the queue (scheduling). These are two separate operations. Enqueueing involves buffer management and discard strategy. A third consideration is the organization of the queue often called the queueing discipline.

Queueing Discipline.

Most routers used in the Internet today implement a single first in first out (FIFO) queueing discipline. This is a simple strategy to implement. It requires no scheduling decision, as the next packet for service is always the one at the head of the queue. However, a FIFO offers no control over the delay a packet will experience in the queue. There is no guarantee of fairness in sharing the link bandwidth between users and no protection against misbehaving users.

Multiple priorities. The simplest extension is to offer multiple priorities with one FIFO queue for each priority. Traffic in a lower priority class is only served if all of the higher priority queues are empty. This allows some simple control over the delay for traffic in the higher priority classes if there is a constraint on the amount of traffic in these classes but there is still no guarantee of fairness or protection from users in the same or higher classes.

Weighted Fair Queuing. In order to offer fair sharing and protection from misbehaving sources, weighted fair queueing has been proposed. This is equivalent to a FIFO queue per class (or per flow) with a scheduler that shares the bandwidth of the link between the classes in a predetermined manner.

Class Based Queuing. We can make things more complicated by introducing the concept of hierarchy to get hierarchical weighted fair queueing or class based queueing. Here, if there is any spare bandwidth left after everyone has been given their predetermined share (because someone is not using all of their allotted bandwidth) then the excess bandwidth is shared according to the rules expressed by the hierarchy.

Finally we can combine all of these queueing disciplines to arrive at a very complex queueing system, for example, several priorities and within each priority level a hierarchy of weighted fair queueing. Whether it is worth all this presumably depends upon how expensive and how scarce a resource the bandwidth is.

Conserving work. The above queueing disciplines will always transmit a packet if there is a packet in the queueing system and bandwidth available on the output link. Only if all of the packet queues serving a link are empty will the link become idle. This class of queueing system is called work-conserving.

There is another class of queueing system that will not necessarily transmit a packet even if the output link is idle. These queueing systems are called non work-conserving. They are used to perform traffic shaping or rate limiting. In such a system each flow can be assigned a specific bandwidth. Packets will only leave the queue if the amount of bandwidth currently consumed by the class to which they belong is less than the predetermined limit.

Discarding Packets.

When a packet arrives at a queueing system there is a fundamental decision to be made. Do I accept this packet or do I discard it? A number of factors affect this decision. First, is there enough room available in the buffer memory? If there is insufficient room something has to be discarded.

The easiest discard strategy to implement is to discard the newly arrived packet. This is called tail-drop. Another approach is to discard packets from the front of the queue until there is enough room now available to store the newly arrived packet. This is the drop from front strategy. Another strategy, typically in a queueing system with per-flow queues, is to drop packets from the longest queues until there is enough room for the new packet. Drop from front and drop from longest queue are reported to result in better performance for TCP.

Discard can be used to control the traffic. Flows can be given a discard preference so that packets from some flows are more likely to be discarded than packets from other flows. Flows can be metered according to some traffic profile. If a flow exceeds its traffic profile the packets can be marked to be discarded in preference to packets from flows that have not exceeded their allotted profile.

Random Early Detection (RED).

Various random discard schemes have been proposed and investigated. RED has been demonstrated to improve performance in large scale networks. It has become accepted and is currently being implemented and deployed. (Some part of this success might be attributed to the choice of acronym which emphasizes the more positive aspects of the algorithm, detection rather than discard.)

TCP interprets lost packets as an indication of congestion. When it detects packet loss it reduces the rate at which it transmits data into the network. Also, under stable conditions TCP gradually increases its transmission rate until it detects packet loss in an attempt to transmit at the highest data rate that the network can support at that time. If a tail-drop algorithm is used this results in the queue at the bottleneck being almost permanently full. Keeping queues full is a bad idea.

Random early detection is an algorithm that attempts to detect the onset of congestion before the queue is full by measuring the average queue length. When it detects that the average queue length exceeds a threshold, for every arriving packet it throws a loaded pair of dice (i.e. computes a random number). The random number determines whether the packet should be discarded. The probability of discard (the loading of the dice) is proportional to the amount by which the average queue length exceeds the lower threshold.

By discarding a packet before the queue is totally full, TCP is instructed to reduce its transmission rate while there is still space in the queue to cope with arriving bursts of packets. By using a random number the available capacity is shared approximately evenly between the competing users. Also the heaviest users are the ones most likely to get a packet discarded because they are sending the most packets. The result of this algorithm is that the queue is kept mostly empty rather than mostly full. So bursts can still be accepted, which is what the queue is for, but TCP still gets its feedback of dropped packets to tell it to back off its transmission rate.

A (Brief) Look at Congestion Control

The algorithm TCP uses to discover the rate at which to transmit data into the network by monitoring dropped packets is loosely called congestion control or congestion management or flow control. Although these terms have (or *used* to have) specific meanings, these days they are roughly synonymous.

The TCP algorithm is also called congestion avoidance even though the algorithm deliberately causes congestion in order to discover the best transmission rate. It doesn't actually manipulate the transmission rate directly, it manipulates a window size, but this is getting far too detailed.

There are other proposals that claim to do a much better job of congestion control. For example, the ATM Forum wrote a specification for Available Bit Rate (ABR) that defines an Explicit Rate algorithm.

While intellectually interesting, the ABR specification basically starts by discarding the entire Internet as we know it, and would require reprogramming the 100 million or so computers attached to it. This approach is not currently seen as a winner. We have to make do with what we've got and improve it slowly.

Buffer Management.

Buffers are where you put the packets when you want to store them. You make a FIFO queue by linking buffers together into a chain. Buffer management refers to the policy and mechanisms by which the buffer resources are controlled.

Buffers are frequently implemented as a shared resource. Many output ports share the same pool of buffers to construct their queues. This is more efficient in its use of memory than replicating an individual pool of buffers for each port. Buffer management must ensure that no single port, or small number of ports, can use up all of the buffers. There must always be a minimum number of buffers available for each port.

Frequently the actual filling, emptying, and chaining of buffers is performed by hardware. Buffer management must control the process of passing control of buffers to the various hardware components. The term buffer management is also often used in a wider sense to include the discard strategy or even loosely in the very wide sense of control of the queueing system.

Scheduling.

Now that we have considered how to put the packets into the queue we get to look at how to take them out of the queue. Keep in mind that taking the packets out is a separate and independent activity from putting them in. When considering the untold complexities of Quality of Service one can lose sight of this fundamental principle.

If the queueing system is a single FIFO then there is no scheduling decision to be made. Always take the packet at the head of the queue. For other queueing disciplines the scheduling decision selects the next packet to be transmitted out of the output port. We discussed examples of scheduling algorithms in the section on Queueing Disciplines.

Quality of Service (QoS).

Last, and definitely the least pleasant, we come to Quality of Service (QoS). QoS is a tar pit. Once you start down that path it is very difficult to stop. And it never ends. Keep in mind **Lixia Zhang's** aphorism: "Quality of service does not generate additional bandwidth." QoS is a concept that comes from circuit switching, from the phone company. It is capitalist switching [3]. For a connection-oriented network it is a measure of the quality of a connection for which, presumably, the customer is paying.

The quality of a connection, or virtual connection, is measured in terms of delay, delay variation, throughput, and loss (or bit errors). Back when all you could ask the network was, "Please give me a phone call," it was pretty easy to measure the quality. But then again, all you could ask for was a phone call — even if you wanted to send images, video, or data.

For the last 20 years or so we have been searching for the pot of gold at the end of the switching spectrum, the grand unified theory of networking, the integrated services network. This is a network of which you can ask anything. You can request a specific throughput, a specific delay, a specific variance of delay, and specific loss characteristics.

Loosely, we call the ability to ask for whatever you want from the network, *quality of service (QoS)*. After 20 years of trying we've discovered that delivering QoS is very, *very* difficult.

It is difficult for many reasons, not the least of which is that the customer does not actually know what he or she wants. And not because the customer is dumb, but because the traffic source cannot be characterized simply. Also because, for a computer, the traffic source is removed from the network by the operating system which inserts its own peculiar characteristics of delay. Further, in the case of the Internet, you can't trust the customer to abide by any declared traffic profile, and if you are going to let the customer request resources you need a method of charging for them.

No doubt we will solve these problems in the fullness of time, at least the technical problems. But right now it would be better to do something simple, for example, economy class, premium, and first class. We may also restrict the access rate of a customer onto and from the Internet, and offer virtual private network tunnels with fixed guaranteed bandwidth.

Policy Based QoS.

We might emulate the frame relay concept of committed information rate across a fixed tunnel. Rather than let the customer ask what they want on a per use basis, control access on a long term basis, or give control to the network manager. This is policy based quality of service rather than per connection based quality of service.

To illustrate the concept of policy based quality of service consider the process of ordering a sandwich on both sides of the Atlantic:

England: "What would you like love, egg or cheese?"

America: "Give me a turkey and pastrami on rye, Monterey jack and cheddar with the works but hold the mustard and pickle."

The user must know what she wants (service), must be able to describe it (traffic characteristics), and must know how to request it (signalling). The English, however, make do with a (simple?) class system and some queueing discipline, where everyone stands in line — according to class of course!

Putting it All Together, Real Fast.

We need to apply hardware acceleration to the packet forwarding operation, to the classification and forwarding, to the switch fabric, and to the queueing system. Depending on the choice of switch fabric, the queueing system and the switch fabric may be integrated into a single component.

Unfortunately it is difficult to combine both the classification function and the forwarding function into a single data structure. So they are typically implemented in separate hardware elements running in parallel.

The classification function requires the equivalent of a content addressable memory search. If a real content addressable memory (CAM) is used there is a restriction on the size of these devices and therefore the number of entries in the classifier. The classification problem is not amenable to an implementation based on hashing because each entry in the table may want to examine a different set of bits in the packet header. Also the classification operation may result in a number of matches, one of which should be selected according to some policy.

The packet forwarding operation requires a longest prefix match lookup on the destination IP address for unicast or the source and destination IP addresses for multicast. Fast implementations of the longest prefix match operation are a subject of current research. The typical software implementation uses a tree structure called a Patricia tree. Some approaches implement this tree structure and the lookup engine in hardware.

Another approach is to reduce the data structure to a very compact representation so that it can be stored in very fast memory, possibly even in the layer 2 cache of a processor. An alternative is to represent the structure in such a way as to reduce the number of memory fetches required per address lookup. A third proposal is to simply use a very, very big table on the grounds that memory is cheap these days.

Problems, Problems. A problem with many of the fast lookup proposals is that they require the forwarding table to be structured into a compact representation. This makes it very difficult, if not impossible, to incrementally insert and delete routes. It is argued that routes do not change very often and only need to be updated once every 30 seconds or so. Some estimates suggest an update interval as long as once every two minutes. Not everyone is convinced that this is acceptable. Two minutes is a long time for incorrect forwarding information to exist, possibly resulting in routing loops.

The design considerations regarding the switch fabric for an IP router are very similar to those for an ATM switch. The additional complication revolves around the fact that IP packets are of variable length. While it is possible to build some forms of switch fabric using variable length packets it tends to be easier to chop the packets into small fixed length data units send them across the switch fabric and then put them back together again. Indeed for some switch fabric designs, e.g. a crossbar, you have to do this or you will obtain poor performance [4]. The problem of reassembly is easier than it is in ATM as there are only a limited number of sources from which concurrently arriving packets have to be reassembled, at most one from each input to the switch fabric.

Buy or Build?

We could discuss the joys of input versus output buffered switch fabrics — but it has all been said before [5]. A more pragmatic approach is to purchase a switch fabric chipset if you can find something at a reasonable price that does vaguely what you want. There are a number of startups working on general purpose switch fabric chipsets with a variety of different sizes and features. Why roll your own if you can get one ready-made at a decent price?

Pain threshold. Always keep in mind the engineering maxim: "Whenever possible use brute force." Only when past the threshold of pain should you try and be clever. Hence a shared memory is popular for modest designs and a crossbar for speeds at which a shared memory becomes challenging. One limitation with a crossbar is that while adding more crossbar chips in parallel can increase the speed of each port, it is difficult to achieve large numbers of ports. So 32 by 32 or maybe 64 by 64 is the limit although one can make each port run as fast as one can afford to pay.

Data Lumps.

Beyond this size there is little choice but a multistage fabric. Things can get messy with a multistage fabric. In a multistage fabric it is possible for all the little data lumps to get out of order (you'll note that I am trying *really* hard not to call them cells). Out of order problems with your data lumps increases the difficulty of reassembly. Finally there is multicast which can be trusted to throw a wrench into any half decent switch fabric design.

Typically queueing will exist in several places in a gigabit router design and for a number of different reasons. Some queueing will be required for pipelining, to permit multiple pieces of hardware to work on a small part of the problem concurrently. The queueing that implements the main queueing system will typically be located on the output port in most switch designs or within the shared memory for a shared memory switch fabric. There is a choice of DRAM or SRAM. SRAM is fast but it is expensive if a large amount of memory is required. DRAM is cheap in large quantities but none too fast.

Perhaps the hardest problem in designing a gigabit router is deciding the amount of flexibility required. Software based routers have a great deal of flexibility in adapting to new functionality and changes in the standards that occur from time to time. Implementing some of the forwarding functions in hardware reduces that flexibility. The tradeoff between flexibility functionality and speed is at the heart of the design problem in the same way that features, time to market and cost are at the heart of the marketing problem.

Marketing Matters.

Gigabit routers come in three flavors: core, edge and enterprise. **Core routers** require high aggregate bandwidth, high packet forwarding rates, and really stable software. They require high-speed interfaces but only a limited number of different types (packet over SONET and ATM at OC-3, OC-48 and soon OC-192). They require a stable implementation of the BGP routing protocol and the ability to handle very large routing tables (several hundred thousand entries). They will probably not require sophisticated queueing and quality of service mechanisms as traffic at the core is concentrated (and bottlenecks are most likely to be at the edge not the core).

Redundancy required. Redundancy is usually supported by using multiple power supplies, duplicating the switch fabric and the controller, and operating the spares as a warm standby. However, anyone that owns a PC will readily appreciate that it is not usually the hardware that goes wrong these days. Hence the emphasis on stable software, especially the routing protocols.

An **edge router** requires less bandwidth and forwarding capacity. Most interfaces are of lower speed but there is a wider range of interface types to support. More sophisticated queueing and quality of service mechanisms are required, as it is very likely that bottlenecks will be found at the edge (high-speed meets low-speed.) This is also where the billing will go when we figure out how to deploy premium services in the Internet. BGP is also required at the edge and so is redundancy.

It is not certain that an **enterprise router** really requires gigabit capacity, at least not yet. Startups that began by aiming at the enterprise have shifted focus to the edge router. However, in the fullness of time we will deploy gigabits here too and maybe sooner than we think. Cost (sorry, price) will obviously be important here so products that are gigabit switches with some simple routing capabilities are likely to be popular.

Additional capabilities such as an integrated firewall or traffic shaping may also become important. Products in this space will have to support a wide variety of interface types and speeds and handle some legacy traffic also.

Conclusion.

As I look back it surprises me that so many of us simply ignored Moore's Law (the functionality that can be implemented in silicon doubles approximately every 18 months). Up until relatively recently IP packet header processing was thought too difficult a problem to implement in silicon. Well it's not, and that changes things some.

Gigabit routers are a recent development so it is not yet clear what set of features are required. Some suggest that simple priority queueing is sufficient whereas others [6] are implementing extremely complex queueing and traffic management capabilities.

So what's the difference between a switch and a router? Not much if you only consider the hardware — just a collection of ASICs and memory. There is much more functionality in the hardware of a gigabit router, but in time, with volume production, the cost of the hardware will fall. The difference lies in the software.

Routing software is very complex and expensive to develop, debug, and test. While there are at present many companies engaged in the development of gigabit routers, some claiming very high aggregate bandwidth, the proof of the pudding is likely to be found in the quality of the routing code. ■

References.

[1] S. Keshav, R. Sharma, "Issues and trends in router design." IEEE Communications Mag., May 1998, p. 144. Also see other articles in this issue.

[2] Henry Reed, "Naming of Parts" <http://www.iprg.nokia.com/~pn/poems/naming_of_parts.htm>, see also T. J. DesJardins, "Assembly of Cells," Proceedings of the ATM Forum Poetry Subworking Group, <http://www.iprg.nokia.com/~pn/atmf_poetry/assembly_of_cells.htm>

[3] P. Newman, "Capitalists, Socialists, and ATM Switching," Data Communications Mag. Dec. 1994 p 126. <http://www.iprg.nokia.com/~pn/papers/datacomm94.html>.

[4] N. McKeown, "A fast switched backplane for a gigabit switched router," Business Communications Review, Dec. 1997.

[5] P. Newman, "ATM technology for corporate networks," IEEE Communications Magazine, Apr. 1992 pp 90-101. http://www.iprg.nokia.com/~pn/papers/switch_design.pdf

[6] V. P. Kumar, T. V. Lakashman, D.Stiliadis, "Beyond best effort: Router architectures for the differentiated services of tomorrow's Internet." IEEE Communications Mag., May 1998, p. 152.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Dr. Peter Newman is a Member of the Technical Staff at **Nokia Sunnyvale**. He was an early employee of **Ipsilon Networks**, where he helped design IP Switching and associated protocols. Before joining Ipsilon Peter helped design an early ATM switch at **Adaptive Corporation**. Peter earned his Ph.D. in fast packet switching at the **University of Cambridge** in 1989. <peter.newman@acm.org>
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~