

ROUTER DESIGN & ROUTE LOOKUP MECHANISMS IN NEXT GENERATION BACKBONE ROUTERS

Muthuvelan KP {kpm@ittc.ku.edu}
Electrical Engineering & Computer Science department,
University of Kansas, Lawrence, KS 66046.

ABSTRACT

With the growth of Internet, the amount of traffic through the backbone networks of Internet Service Providers (ISP) is enormous. As more machines are being connected to the Internet and new applications are being developed, the rate of growth is on the increase too. This has raised a need to build scalable, reliable and faster routers at the backbone networks. In the backbone routers the number of routes in the forwarding table is very large. With very high speed links being deployed in the backbone networks, route lookup for the data traffic is very key aspect. This paper gives a brief insight into the architectural evolution of backbone routers from the perspective of forwarding, queuing, switching and scheduling. The paper mainly discusses various route lookup mechanisms being currently used in backbone routers. It discusses various solutions like Binary trie, Path Compressed trie, Level Compressed (LC) trie, etc. It gives an introduction to various advanced techniques like prefix expansion, disjoint prefixes, multi-bit tries, etc. The paper also discusses two categories of hardware based solutions. Firstly, ASIC based solution involving pipelines architecture. And it discusses solutions based on CAM (Content Addressable Memory). The paper finally discusses about the future of fast route lookup mechanism in backbone routers.

KEY WORDS

router architecture, queuing, route lookup, trie, forwarding, fast ip lookup, CAM

1. INTRODUCTION

The key functions of router can be classified into routing and forwarding. Routing is the process of running various protocols like BGP, OSPF, LDP, etc to arrive at set of efficient routes towards various network destinations. Forwarding refers to the process of receiving packets, performing route lookup on the packet and sending the packet on an output interface. Forwarding involves various other functions like policing, rate-shaping, QOS, etc.

A backbone router is different from any other low-end router. It performs forwarding at very high speeds and does route lookup on large number of routes ($\approx 120,000$). In backbone routers, routing protocol traffic and data traffic follow different paths through the router. These are usually called the slow-path and fast-path respectively. Some of the other issues to be considered while designing a backbone router include performance, reliability and scalability. The performance of a router includes

- Throughput – rate at which packets sent/received without loss.
- Latency – time spend by a packet inside the router.

Backbone routers have a very high throughput and very low latency time. Reliability is the ability to be tolerant to failures of various components and it is achieved mostly by redundancy.

The following sections would give a brief introduction to router architectures followed by a survey of various route lookup mechanisms.

2. ROUTER ARCHITECTURES

The general architecture of a router consists of a set of line cards with varying interfaces and memory to store the incoming packets. In early router design, both the data traffic and the control traffic goes through the same path inside the router (figure 1). All the packets received by the router goes through the shared bus and is stored in the shared memory. The central processor processes the packets and sends out. This design did not scale well because of the single processor and the bandwidth limitation of the shared bus.

In present generation routers, instead of having a single central processor, multiple processors are located (one or more on each line card). Each processor performs various functions on the packets received on that line card. Instead of having a shared bus as a switching medium, a number of alternative technologies are available. The following sections will briefly describe the various alternatives available for various aspects of router design.

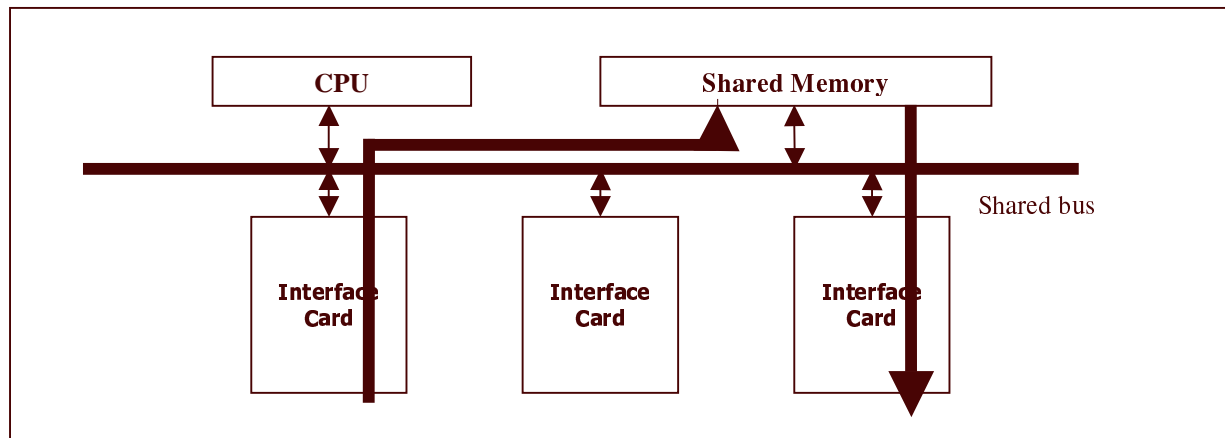


Figure 1. First generation router architecture

Queuing – A router is considered a “store and forward” device. The packets received by a router are stored inside the router before being forwarded. Since the router should preserve packet ordering, the packets are stored in the form of various queues. There are broadly two types of queuing [5] – input queuing and output queuing.

In Input queuing, the packets are queued based the input interface through which it is received. The advantage of input queuing is that it prevents backplane speed (explained later). The disadvantages associated with input queuing are Head-of-Line (HOL) blocking and difficulty in QOS characterization. The HOL blocking is where a packet is blocked by another packet ahead which is waiting for the output port to be available to it (figure 2). The other limitation of input queuing is in characterizing QOS requirements. Most of QOS specifications are specified in terms of outgoing ports. Hence, if input queuing is used, the input port has to maintain QOS requirements of all outgoing ports.

Output queuing is when all the packets are queued based on the outgoing ports. Output queuing removes all the limitations of input queuing. But the problem with output queuing is the “backplane speedup”. If packets received on all the ports of a router decide to go through a particular output port then the back plane needs bandwidth equal to the sum total of all the ports in the router. This is known as *backplane speedup*. Even with this limitation output queuing is more preferable than the input queuing.

There is another form of queuing which combines the best of both input and output queuing. It’s known as Virtual Output Queuing (VOQ). In this case, the packets have multiple queues at each input port for various output ports.

Switching – It is the process of moving packets from one line card to another within the router. There are number of design alternatives for switch fabric. The first generation routers used a shared memory model with a single processor. This had the single processor as the bottleneck. The later routers used a shared bus with independent processors on the line card performing packet transfers using DMA. The limitation with this approach is the bus arbitration overhead. Presently routers use crossbar or point-to-point links as switch fabric.

Packet Scheduling – The packets which are stored in the queues has to be scheduled properly and transmitted through the output ports. The common scheduling algorithms are First Come First Serve (FCFS) and priority scheduling. Both have their own inherent limitations. The most common packet scheduling algorithm used is Weighted fair Queuing (WFQ). According to this scheme each packet stream is allocated a weighted percentage of the output bandwidth.

3. ROUTE LOOKUP MECHANISMS

Route lookup is one of the key aspects of a backbone router for various reasons. Firstly, the number of routes in backbone networks is very large. Secondly, the rate at which router lookup is performed is very high. The backbone routers are connected to high speed links and support very high throughput and very low latency.

The advent of CIDR [1] (Classless Inter Domain Routing) has made route lookup more complex. Before CIDR, in classful routing each prefix had a predefined mask associated with it and route lookup was easy. But with classful addressing the address

space was depleted at a very fast rate and lots of address space was wasted. With CIDR, each prefix

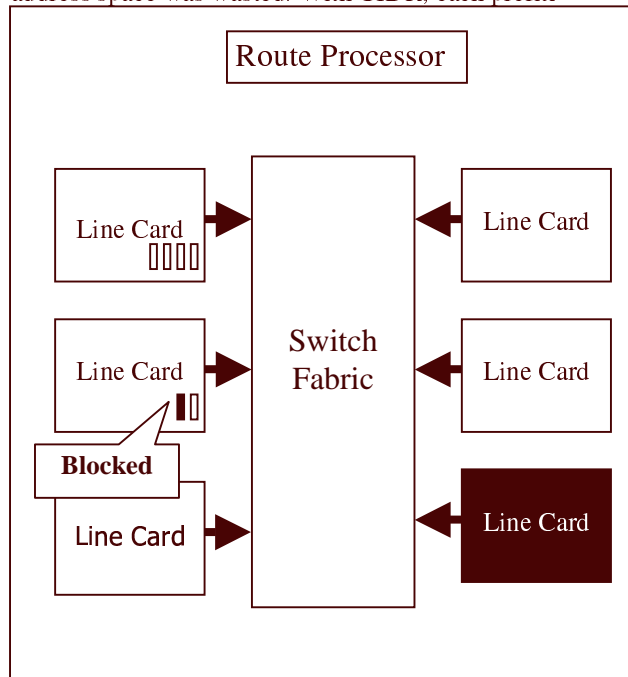


Figure 2. Head of Line Blocking

has a network mask associated with it. And while trying to find route for a packet, the longest matching prefix (more specific) was chosen.

Any route lookup mechanism has an optimized data structure to store the prefix information and a corresponding route lookup algorithm. Some of the key issues to be considered while designing the route lookup mechanism are

- **Memory accesses** – The number of memory accesses determines the time to perform route lookup. The more the number of memory accesses, the more the route lookup time. The variance in the number of memory accesses should also be minimal. In other words, the number of memory accesses should be nearly the same for different lookups.
- **Incremental updates** – The forwarding tables of the routers keep changing constantly. The addition and deletion of prefixes to the data structure should be faster.
- **Memory requirement** – The amount of memory required to store the prefixes should be less.

Route lookup algorithms are commonly classified into two categories – search by values and search by length. The search by values technique includes hash, array, etc., and the time to perform route lookup is dependent on the number of routes present. The search by length techniques are trie-based solutions, and the time to perform route lookup is dependent on

the length of the prefix being matched. Most common solution to do route lookup is using tries and variations of it.

Classical Solution – Binary trie

A binary trie-based route lookup is one of the most common solutions. A binary trie is constructed from the set of routes (or prefixes) to be added to the forwarding table (figure 3). The nodes of the trie, which indicate prefixes, are separately marked as such. The destination IP address of the incoming packet is used as the key to search the binary trie.

Search mechanism – Each bit of the destination IP address, starting from the left, is used to traverse the trie. While traversing the trie, the current BMP (Best Matching Prefix) is remembered. The search ends when no matching branch can be found, and the BMP remembered so far is the result of the route lookup. In binary search, the search space reduces by half at each stage, and the nodes do not store any prefix information.

Update operation – When new prefixes are to be added, the binary trie is searched for the prefix and inserted if not present. Deletion operation involves just unmarking the corresponding node.

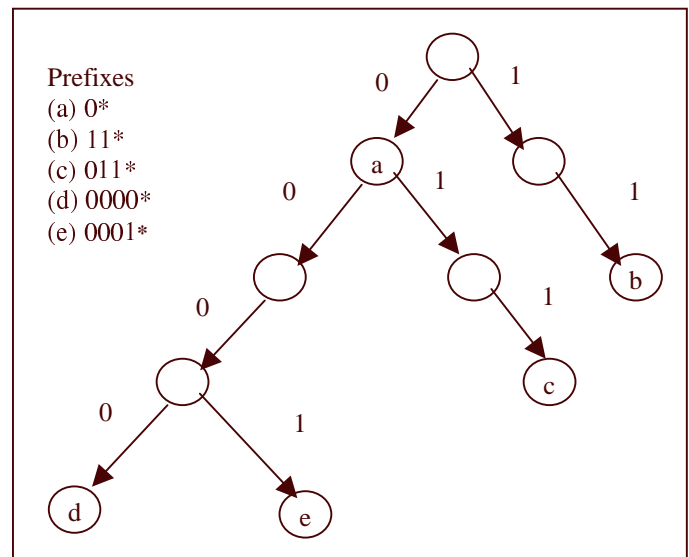


Figure 3: Binary Trie

Path-Compressed trie

In sparsely populated binary tries, long sequences of single-child nodes may exist. In this case, all the bits are inspected even though no real branching decision is made at those nodes. This increases both the memory space required and the number of memory accesses. A path-compressed trie is one in which all

the single-child nodes are collapsed. Each node has a bit number field to match. While descending the trie we may skip some bits. Hence, a node marked as a prefix also stores the prefix information explicitly. Each bit is matched against the bit number specified in the nodes of the trie. The search ends when no branch can be taken. The update operations are slightly more involved than that of a binary trie.

There are several variations of path-compressed tries. The two most commonly used are the one proposed by Skowler [6] and the BSD trie. In Skowler's method, the BMP is remembered while descending the trie. While in the case of BSD trie, the last node backtracks to the BMP.

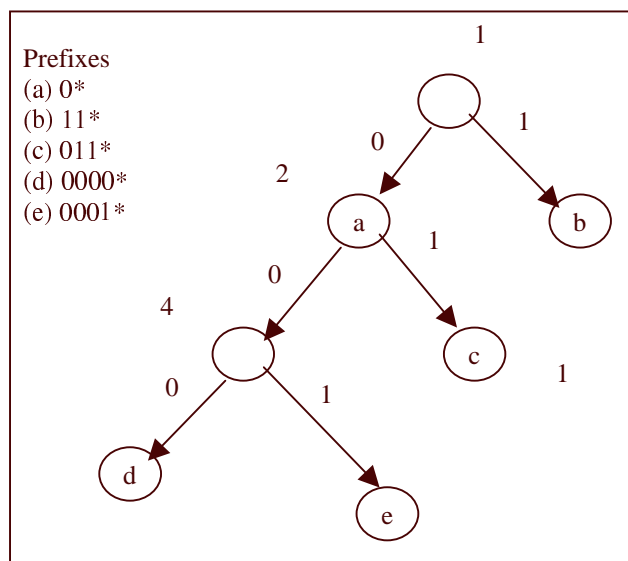


Figure 4. Path Compressed Trie

Advanced Techniques

Prefix Expansion – It is the process by which a prefix is expanded to an equivalent set of longer prefixes. During expansion if there is a collision with an already existing longer prefix then the expanded prefix is dropped. Consider prefixes $0^* \rightarrow a$ and $01^* \rightarrow b$, where a and b are next hop information. The prefix 0^* can be expanded into equivalent prefixes of length two as 00^* , 01^* , 10^* and 11^* . Since there is already a prefix $01^* \rightarrow b$, the prefix $01^* \rightarrow a$ is dropped. The final set of prefixes are $00^* \rightarrow a$, $01^* \rightarrow b$, $10^* \rightarrow a$ and $11^* \rightarrow a$. This technique is used in construction of multibit tries (discussed later).

Disjoint Prefixes – In a given set of prefixes, one prefix can be a prefix of another prefix. Hence, the longest matching prefix has to be found. One way to avoid longest matching rule is to translate a set of

prefixes into a set of disjoint prefixes. A set of disjoint prefixes is one in which no prefix is a prefix of another prefix.

A trie with disjoint prefixes can be obtained from a binary trie by adding leaves to nodes which have single child and unmark the internal nodes. This is also referred to as *leaf-pushing*. A trie with disjoint prefixes has prefix information only at the leaves and not in the internal nodes.

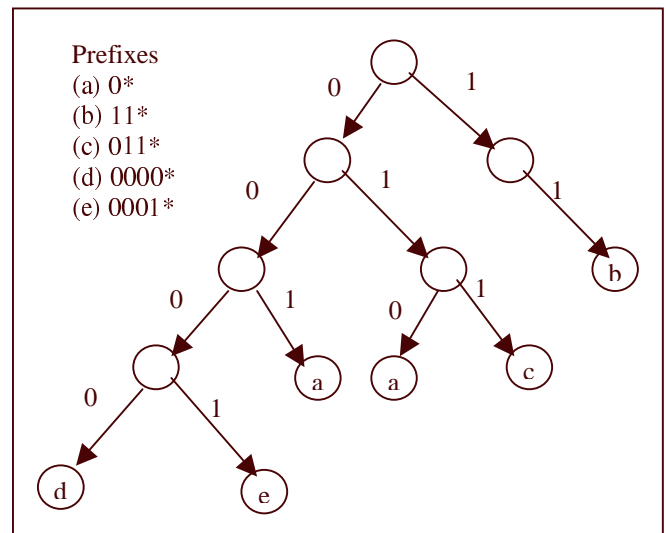


Figure 5. Disjoint Prefixes.

Multibit Tries

In the case of a binary trie, we inspect one bit at a time. Hence, for IP addresses, in the worst case, it would take 32 memory accesses. If we can inspect more than one bit at a time, then we can reduce the number of memory accesses. If 4 bits are inspected at a time then only 8 memory accesses are required at the maximum. A multibit trie is one in which more than one bit is inspected at a time. The number of bits inspected at a time is called the *stride*. If all the nodes at the same level have same stride size then its called a fixed stride trie else is called variable stride trie. Multibit tries cannot support arbitrary prefix lengths. In order to construct a multibit trie from a standard set of prefixes, prefix expansion has to be done based on the chosen stride. The searching mechanism for multibit bit tries is same as binary tries except that more than one bit has to be inspected at a time. Multibit tries also does linear search on the prefix lengths like the binary tries, but it is faster as it takes larger strides.

Choice of strides – The size of the stride decides the number of memory accesses required and the space consumption. If the stride size is large, then the number of memory accesses is less, but the space

required by the multibit trie is large. The structure of the trie can be used to decide on the stride size. If a subtrie of level n is fully present, then it can be replaced by a multibit subtrie of stride n .

Update operations – Since, multibit tries are constructed by performing prefix expansion, each subtrie has a local BMP (Best Matching Prefix). Hence, update operations are limited to a particular subtrie. Deletion has a problem with multibit tries. While performing prefix expansion, a set of longer prefixes could have overwritten a set of expanded prefixes. If the one of the longer prefixes is deleted, it has to be replaced by the expanded prefix which is not present in the trie. Hence, the original prefixes have to be stored separately.

Multibit Disjoint Tries

In the case of multibit tries the internal nodes store prefix information. Hence, the memory consumption of multibit tries is large. In multibit disjoint tries, the internal nodes do not store prefix information. The memory consumption is less compared to multibit tries. But the major problem with multibit disjoint tries is in update operations. Due to leaf pushing, each subtrie does not have a local BMP and update operations are time consuming and difficult.

Level Compressed (LC) Tries

Level Compressed tries are path-compressed multibit tries proposed by Nilsson [8]. In a binary trie, a full binary subtrie is replaced by a single level multibit trie. This operation is performed recursively over the binary trie. It also removes single child links. Since multi-level binary subtrie is being replaced by a single level multibit trie its known as Level Compressed trie. In this method the structure of the binary trie guides the construction of LC trie. The decision to replace a binary subtrie with a multibit subtrie is based on a fill factor k ($0 < k \leq 1$), which is user configurable. The fill factor is the percentage of nodes present in a binary subtrie. The common value used is 0.5. In order to save memory, the trie is stored as a array, with first level nodes first, second level nodes next, etc. Moreover the internal nodes in the trie do not have any prefix information. The leaves of the trie have a list of prefixes for that path. Performing incremental updates in this method is very difficult.

Binary search on prefix length

All the tries discussed so far do sequential search on the prefix length. Waldvogel *et al*[13] proposed a method to do binary search on the prefix length

without using trie data structure. In this method, there is a separate hash table for prefixes of each length. If a match is found in hash table of particular length, hash table of higher prefix length is looked up. But if a match is not found in a hash table, then the prefix could be found either in hash table of lower length or higher length. In order to guide the search in cases like that fictitious prefixes are added to hash tables of lower lengths. For prefix 11111^* found in hash table of size 5, prefixes 1^* , 11^* , 111^* and 1111^* are present in hash tables of lower lengths. These fictitious prefixes are called *markers*. The markers could mislead the search. With above marker it could mislead the search for prefix 11110^* . Markers occupy considerable space in a sparsely distributed trie.

The complexities of the various algorithms discussed so far are tabulated in figure 6. The number of prefixes is indicated by ‘N’ and the prefix length is ‘W’.

Scheme	Worst case lookup	Update	Memory
Binary trie	$O(W)$	$O(W)$	$O(NW)$
Path Compressed Trie	$O(W)$	$O(W)$	$O(N)$
Multibit trie (stride = k)	$O(W/k)$	$O(W/k + 2k)$	$O(2kNW/k)$
LC trie	$O(W/k)$	-	
Binary search on prefix lengths	$O(\log_2 W)$	$O(N \log_2 W)$	$O(\log_2 W)$

Figure 6. Complexity Chart

Hardware Approaches for Route Lookup

The route lookup algorithms discussed so far are implemented in processors located on line cards of the backbone routers. There are other hardware solutions for performing fast route lookup

ASIC based solution

Instead of using generic processors for performing route lookup functions, custom ASICs [9] (Application Specific Integrated Circuits) are designed for such purpose. ASICs have finer control over memory access and can be designed with pipelined architecture (figure 7).

Content Addressable Memory (CAM)

Content Addressable memories [14] are inverse of RAM, it performs a parallel search on the contents of

memory and returns a result. CAMs are useful for performing lookup on flat address space. Using CAM for hierarchical addresses like IP addresses involves

matched against the CAM. Binary search on prefix priority is performed.

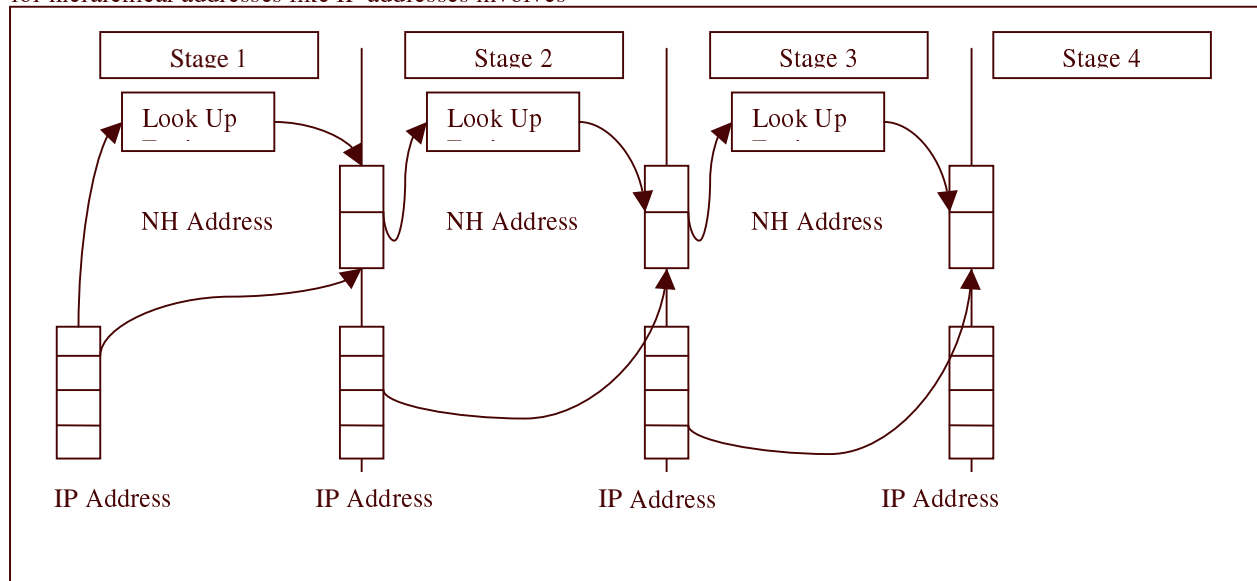


Figure 7. Pipelined Architecture

additional logic. There are 2 types of CAM available – binary and ternary CAM. In binary CAM 0's and 1's can be stored. In the case of ternary CAM, 0's, 1's and don't care bits can be stored. The following are some of the solutions using CAMs.

Binary CAM

Multiple Cycle, Single Logical CAM – In this method, there is mask register surrounding the CAM. When a packet arrives the mask register is loaded with highest mask length and searched. If no match is found lower mask length are tried.

Single Cycle, Multiple Logical CAM – In this method there are multiple CAMs storing prefixes of each length. There are prioritizer logic surrounding all the CAMs so that match with highest mask length is returned as result.

Ternary CAM

Single Cycle, Single Logical CAM – In this case a CAM which searches the memory in ordered way is used. The prefixes are stored with longest prefix stored first. Major limitation of this method is that, update operation require total reinitialization of the CAM.

Multiple Cycle, Single Logical CAM – In this case the prefixes are stored with explicit priority information. The longest prefix is given the highest priority. when a packet arrives <prefix, priority> is

4. CONCLUSION

The router architectures are evolving to meet the growing needs of the Internet. Building faster, denser and reliable routers is dream pursued by lots of people. Route lookup algorithms are now being reexamined to perform route lookup on various new forms of traffic. Lookup on MPLS, VPN and multicast traffic require additional functionality. MPLS has flat address space of 20 bits, VPNs have multiple forwarding table corresponding to each customer and multicast traffic require lookup on both source and destination address. Integrating route lookup of various forms of traffic in faster and scalable form is the challenge being pursued presently.

REFERENCES

- [1] Y. Rekhter, T. Li, An Architecture for IP Address Allocation with CIDR, RFC 1518, September 1993.
- [2] James Aweya, IP Router Architectures: An Overview, Nortel Networks, Ottawa, Canada, 1999.
- [3] Shivkumar Kalyanaraman, High Speed Router Design, Rensselaer Polytechnic Institute, 2001.

- [4] Masatoshi Kumagai, Satoshi Nojima & Hiroshi Tomonaga, IP Router for Next-Generation Network, Fujitsu, 2001.
- [5] David Hakaia, Router Architecture and Switching Performance, Cisco Systems, 2000.
- [6] K. Sklower, "A Tree-Based Packet Routing Table for Berkeley Unix", *Proc. 1991 Winter Usenix Conf.*, 1991.
- [7] S. Nilsson & G. Karlsson, Fast Address Look-Up for Internet Routers, *Proceedings of IEEE Broadband Communications 98*, April 1998.
- [8] S. Nilsson & G. Karlsson, IP-Address Lookup Using LC-Tries, *IEEE Journal on Selected Areas in Communications*, June 1999.
- [9] William Eatherton, ASIC Based IPV4 Lookups, 1998.
- [10] E. Will, Hardware-Based Internet Protocol Prefix Lookups, Washington University Electrical Engineering Department, MS thesis, 1999.
- [11] David E. Taylor, John W. Lockwood, Todd S. Sproull, Jonathan S. Turner & David B. Parlour, Scalable IP Lookup for Programmable Routers, *IEEE Infocom*, 2002.
- [12] V. Srinivasan & George Varghese, Faster IP Lookups using Controlled Prefix Expansion, *Proceedings of ACM Sigmetrics, Sep 98 & ACM TOCS 99*.
- [13] M. Waldvogel, G. Varghese, J. Turner, & B. Plattner. Scalable high-speed prefix matching, *ACM Transactions on Computer Systems*, 2001.
- [14] A.J. McAuley and P. Francis, "Fast Routing Table Lookup Using CAMs," *IEEE INFOCOM*, 1993.