

# Shadow IP Route Caching for Trusted Internet Routing

Rama Sangireddy

Dependable Computing and Networking Laboratory

Department of Electrical & Computer Engineering

Iowa State University, Ames, IA, 50011, USA

*sangired@iastate.edu*

**Abstract**—Routers forward IP packets based on the destination address lookup guided by the longest prefix matching algorithm. Destination address of each IP packet is compared against all the prefixes in routing table and the packet is forwarded to a next hop port (NHP) associated with the longest of all matched prefixes. A wide scale effort is on to provide fast address lookup schemes for routing packets at wire speeds. The set of prefixes in a routing table is obtained by aggregation of all routes gathered by backbone router guided by the border gateway protocol (BGP). An announcement of a false route by an autonomous system (AS) results in an undesired prefix, or an unwarranted NHP for a prefix in the routing table. Such announcement of a false route can occur due to an operational fault or an external malicious (intentional) attack. Packets forwarded based on such false routing information will either be dropped, or forwarded to a node predetermined by the external attacker, or can be made to constantly circulate between a set of nodes thus creating an explosion in traffic for the nodes. Our past work includes both hardware and software based efficient address lookup mechanisms. This paper introduces *shadow route caching*, a mechanism to detect and avoid forwarding packets to bogus routes. Our scheme enables the fast address lookup mechanisms to perform intelligent packet forwarding to provide a trusted Internet routing environment.

## I. INTRODUCTION

**T**HE effective handling of the tremendous amount of Internet traffic and its continuous doubling every few months depends on the efficacy of the routers. One of the key issues in router performance is the IP routing lookup mechanism used for the ferrying of the large number of incoming communication packets to respective outgoing links. Since the introduction of Classless Inter Domain Routing (CIDR) in 1993, the IP address lookup mechanism has been designed based on the longest prefix matching (LPM) algorithm. The continuous increase in the number of users on the Internet is causing expansion of routing tables. This increases the search space of prefixes against which the destination address of each packet needs to be matched. The fast increasing transmission link rates too

place higher demands on the performance of routers. Besides, with the advent of IPv6, the IP address length used for matching the prefixes is going to be 128 bits, as against the 32-bit long address in IPv4. In these circumstances, where IP routing tables are expanding in both the dimensions, i.e., address length and number of prefixes, the routing mechanisms developed should address scalability with greater importance.

The set of prefixes in a routing table at a node is obtained by aggregation of all routes gathered by backbone router guided by the border gateway protocol (BGP). The Border Gateway Protocol (BGP) is an inter-autonomous system routing protocol. An autonomous system is a network or group of networks under a common administration and with common routing policies. BGP is used to exchange routing information for the Internet and is the protocol used between Internet service providers (ISP). BGP neighbors exchange full routing information when the TCP connection between neighbors is first established. When changes to the routing table are detected, the BGP routers send to their neighbors only those routes that have changed. BGP routers do not send periodic routing updates, and BGP routing updates advertise only the optimal path to a destination network. An announcement of a false route by an autonomous system (AS) results in an undesired prefix, or an unwarranted NHP for a prefix in the BGP routing table. Packets forwarded based on such false routing information will either be dropped eventually, or forwarded to a node predetermined by the external attacker for malicious reasons, or can be made to constantly circulate between a set of nodes thus creating an explosion in traffic for the nodes. Zhao *et al.* [1] have presented a protocol enhancement to BGP which enables BGP to detect bogus route announcements from false origins. Besides such enhancements in the BGP protocol, protection mechanisms at IP packet forwarding level are required to provide higher degree of security in Internet routing.

Earlier, we presented a reconfigurable hardware solution [2], using the well received concept of Binary Deci-

sion Diagrams (BDDs), that provides a high-speed IP address lookup and enables a data throughput of 200 Gbps (average packet size of 1000 bits) in the current day large routers. We had also proposed software based solutions involving two algorithms - *Elevator – Stairs* algorithm and *logW – Elevators* algorithm [3], for fast lookups in large routing tables. Both analytical and experimental results demonstrate that our algorithms are capable of handling high packet rates. For AADS router with 33,796 prefixes, the *Elevator – Stairs* algorithm gives an average throughput of 15 Million lookups per second (Mlps) with a memory consumption of 517 KB, and the *logW – Elevators* algorithm gives an average throughput of 20.5 Mlps with a memory consumption of 1413 KB. In this paper, we introduce *shadow route caching*, a mechanism in supplement to the fast address lookup schemes, to detect and prevent forwarding of packets to false routes.

The rest of the paper is organized as follows: Section 2 contains a brief description of the longest prefix matching problem. In Section 3, we discuss related research. Section 4 gives details of our work on fast address lookup schemes. In Section 5, we introduce our proposed mechanism to provide a trusted Internet routing environment. Section 6 concludes the paper.

## II. LONGEST PREFIX MATCHING

The routing of communication packets in the IP domain is done on the Next-Hop basis. The packet reaches its final destination in multiple hops. The next hop for a packet is determined by the router based on its destination IP address. Each router has a database, in the form of a routing table, of prefixes of varying length and the corresponding next hop port (NHP) for each prefix. A typical routing table is shown in Table I.

TABLE I  
A SAMPLE ROUTING TABLE.

<i>Prefix</i>	<i>length</i>	<i>NHP</i>
*	0	Z
0*	1	A
01*	2	C
10*	2	B
001*	3	A
101*	3	B

The length of the prefixes can vary from 0 to 32 bits in IPv4, and from 0 to 128 bits in IPv6. For an incoming packet, its destination address is compared with all the current prefixes in the routing table and

the NHP associated with the longest matching prefix is determined to be the output port for the packet. For an example shown in Figure 1, a destination IP address 129.186.200.205 matches three prefixes 129/8 (prefix/length), 129.186/16 and 129.186.192/20, the longest matched prefix 129.186.192/20 is considered to be the best match and the packet is routed to the output port associated with that particular prefix. In other words, routing based on the longest prefix matching is equivalent to routing the packet to a location as close as possible to the destination. If none of the prefixes match with the destination IP address, the packet is sent to a default port, which is associated with a prefix of length zero.

The metrics considered in designing IP lookup algorithms are *Preprocessing time*, *Storage requirement*, *Lookup rate* and *Update time*. Lookup rate is the most significant parameter followed by update time. With the latest advancements in network technology, communication speed is leaping from 10 Mbps Ethernet to 100 Mbps Fiber Distributed-Data Interface (FDDI) to gigabit Ethernet. With the OC192c Line (Line-rate 10Gbps), 31.25 million packets (average size of 40Bytes) have to be processed each second, while for the OC768c (Line-rate 40Gbps), the processing rate required is 125 million packets per second. Besides, the large IPv4 routing tables known today typically contain around 50,000 prefixes and a large IPv6 routing table is expected to contain around 500,000 prefixes. Consequently, the need for enormous amount of data processing at phenomenal speeds, based on longest prefix matching in a large database of prefixes, makes the problem more complicated. Another major problem is frequent updating of the routing table [4]. As the topology of the network changes, new routing information is distributed among the routers resulting in changes in their routing tables. Consequently, one or more entries must be added, updated, or deleted from the table. Only a routing mechanism with a good combination of complexities for lookup, update and memory requirements can perform well in accordance with future trends in Internet routing.

## III. RELATED RESEARCH

The IP routing lookup schemes introduced so far can be broadly classified into two categories, viz., *software* and *hardware* approaches [5]. In amelioration to the classical binary trie traversal approach, several software solutions have been proposed. One of the first was the prefix matching algorithm using *path-compressed tries* [6] based on the PATRICIA (Practical Algorithm to Retrieve Information Coded in Alphanumeric) trie introduced in 1968 [7].

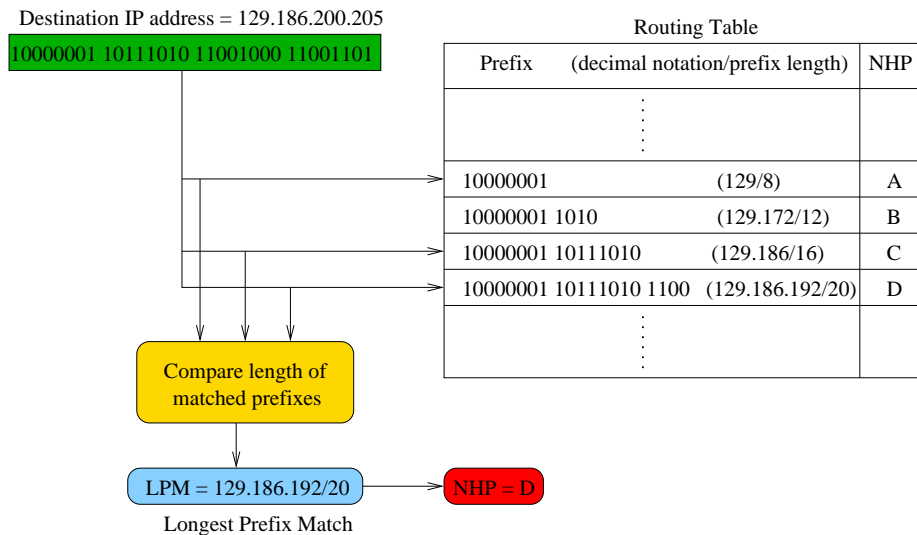


Fig. 1. Packet routing based on longest prefix matching mechanism.

The other schemes subsequently proposed include the various trie based approaches [8], [9], [10] and other binary search methods like *binary search on prefix lengths* [11], [12] and *multiway and multicolumn search* [13]. Besides, other schemes based on *prefix expansion* [14], *string matching* [15] and *level compression tries* [16] have been proposed. Apart from the software based schemes, attempts have been made to design hardware mechanisms for prefix matching to enable high-speed routing. Various hardware schemes like Content Addressable Memories (CAMs) [17], memory lookup based schemes [18], [19], CPU caching [20], and circuit logic implementation using binary decision diagrams (BDDs) [2] have been proposed. Pao *et al.* proposed a hardware architecture [21] implemented with the partition of binary trie into multiple levels, and Taylor *et al.* proposed a reconfigurable device based Fast Internet Protocol Lookup (FIPL) engine [22] for high-speed routing. Recently, Zhao *et al.* [1] have presented a protocol enhancement to BGP which enables BGP to detect bogus route announcements from false origins. This provides a mechanism to provide security at the route aggregation and building of the routing table.

#### IV. FAST IP ADDRESS LOOKUP SCHEMES

##### A. BDD based hardware IP address lookup engine

The proposed scheme is motivated from two observations, first being that even at the largest Network Access Point, the number of next hop ports (NHPs) is generally not greater than 256. Hence, a next hop port (NHP) associated with any prefix in the routing table can be encoded using a 8-bit binary code. For example, any next hop port

(NHP) in the MAE-east [4] routing table can be safely represented by a 6-bit binary code. Every bit of the output port can be computed by a combinational logic circuit whose optimal minimization is obtained with the help of Binary Decision Diagrams. The second observation is that the number of *effective nodes*, defined as the minimal number of nodes required to construct a binary decision tree in order to cover all the prefixes in the routing table, is significantly smaller as compared to the upper bound on the theoretically required number of nodes. It is observed that, for the 32-bit IP address with the biggest available routing table of MAE-West [4], the number of redundant nodes is more than 99.99%. Thus constructing the binary decision tree with a fewer nodes and without any redundant nodes makes it very attractive for the application of Binary Decision Diagrams to optimize the logic. A detailed discussion on the BDD based address lookup scheme and its implementation in a reconfigurable platform can be found in [2].

The implementation of the routing scheme shows that the BDD hardware engine gives a throughput of up to 172.1 Million lookups per second (Mlps) for a large MAE-east routing table with 24,792 prefixes, a throughput of up to 168.6 Mlps for an MAE-west routing table with 29,487 prefixes, and a throughput of up to 229.3 Mlps for the Pacbell routing table with 6,822 prefixes. Thus, a data throughput of 200 Gbps (with an average packet size of 1000 bits) can be obtained in the router implemented with the BDD based hardware address lookup engine.

We measured the performance of our scheme with prefix database of real-time snapshots of various routing tables [4]. The implementation of the routing mechanism is

performed as discussed earlier. The lookup time is measured as the propagation delay between the input and output ports of the combinational logic. This is same as the time taken for signal propagation along the critical path between the input and outputs of the logic. The critical path exists between one of 32 input signals and one of eight outputs. Thus, this measurement of propagation delay gives the worst-case lookup time. The worst-case lookup time and the corresponding packet throughput for the proposed scheme, for different routers, are shown in Table II.

TABLE II

LOOKUP TIME PERFORMANCE ANALYSIS OF BDD BASED ROUTING ENGINE. THROUGHPUT IS NUMBER OF PACKETS PER SECOND.

<i>Router</i>	<i>Prefix count</i>	<i>lookup time (ns)</i>	<i>throughput (Million)</i>
MAE-west	29487	5.93	168.63
MAE-east	24792	5.81	172.11
AADS	33796	5.69	175.74
PacBell	6822	4.36	229.35

Further, in this section we present two high-speed routing algorithms, *Elevator-Stairs* algorithm and *logW-Elevators* algorithm, both software based approaches for fast lookups in large routing tables, which were discussed in detail by Futamura *et al.* in [3].

### B. Software based approaches

1) *Elevator-Stairs Algorithm*: A binary trie is a tree consisting of a set of bit strings  $S_1, S_2, \dots, S_n$  where each edge is labeled with exactly one bit. The space required for a binary trie is  $\mathcal{O}(NW)$  where  $N$  is the number of strings and  $W$  is the maximum length of a string. The space can be reduced to  $\mathcal{O}(N)$  by compressing each maximal non-branching path into one edge, as shown in Figure ???. Such a tree is called PATRICIA trie [7]. Searching the longest matching prefix for an IP address using PATRICIA trie takes  $\mathcal{O}(W)$  time in the worst case regardless of the number and lengths of the prefixes. This run time complexity can be improved by leaping multiple bits at once. The *Elevator-Stairs* algorithm uses hash tables with multiple bits as a key to skip multiple levels of the PATRICIA trie. A data structure is created so that the search algorithm can jump  $k$  levels (or string depth) of PATRICIA trie, where  $k$  is an integer between 1 and  $W$ . Let this structure be called the  $k^{th}$ -level-tree. Let  $k$ -level of a PATRICIA trie denote a level (string depth)  $ik$  for some integer  $i$  such that  $0 \leq i \leq \lfloor W/k \rfloor$ . The building of the  $k^{th}$ -level-tree from the PATRICIA trie is described in detail in [3].

The search for longest matching prefix on an IP address  $p$  using  $k^{th}$ -level-tree starts at its root and sets the variable *current\_port\_number* to be the default port number. The variable *current\_port\_number* stores the port number assigned to the longest matching prefix of length less than the string depth of the current node. At string depth of  $ik$  in the  $k^{th}$ -level-tree, the search algorithm updates the *current\_port\_number* to the port number stored in the current node in the  $k^{th}$ -level-tree, if the node has a copy of the port number. Subsequently, the search algorithm checks the hash table to see if a node with the key  $p[ik + 1 \dots (i + 1)k]$  exists. If such a node exists, the lookup mechanism follows the pointer to the node at level  $(i + 1)k$  and continues the search. If such a node does not exist, it indicates that the search for longest matching prefix must end in the PATRICIA trie between levels  $ik$  and  $(i + 1)k$ . Hence, lookup follows the pointer to the corresponding node in the PATRICIA trie and performs regular PATRICIA trie traversal which gets completed within  $k$  node traversals.

Figure 2 shows two cases of LPM search, one at level  $3k$  and the other at level  $2k + 2$  from the root node. In the former case, the search finds successive matches in the hash tables at levels  $k, 2k$  and  $3k$ , but fails to find a match at level  $4k$  and also fails to traverse the PATRICIA trie. Thus an inference is drawn that the LPM for the IP address is in level  $3k$  and subsequently the NHP information is retrieved from the corresponding node in the hash table. In the latter case, the search finds a match in hash tables at levels  $k$  and  $2k$ , but fails to find a match in level  $3k$ . Subsequently the search traverses the PATRICIA trie starting from the corresponding node in level  $2k$ , and finds a match at a node in level  $2k + 2$ . Thus, in the worst case, the search goes through  $\lfloor \frac{W}{k} \rfloor$  levels of the  $k^{th}$ -level-tree and  $k - 1$  node traversals in the PATRICIA trie. Going through each level takes constant time provided hash table lookup takes constant time, and hence the total search time is  $\mathcal{O}(\frac{W}{k} + k)$ , which is minimized when  $k = \mathcal{O}(\sqrt{W})$  and achieves a search time of  $\mathcal{O}(\sqrt{W})$ .

2) *logW-Elevators Algorithm*: IP address lookup can be performed in  $\mathcal{O}(\log W)$  time using a memory space of  $\mathcal{O}(N \log W)$ . It is achieved by having  $k^{th}$ -level-trees for  $k = \frac{W}{2}, \frac{W}{4}, \dots, 2$ , in addition to the PATRICIA trie. Thus the total space required is  $\mathcal{O}(N \log W)$  since each  $k^{th}$ -level-tree consumes  $\mathcal{O}(N)$  space. If the root node of  $\frac{W}{2}$ -level-tree has a pointer to a node with key  $p[1 \dots \frac{W}{2}]$ , such a path exists in PATRICIA trie and the search reaches the node that is indicated by the pointer. It is also possible to determine if the longest prefix match is already found at

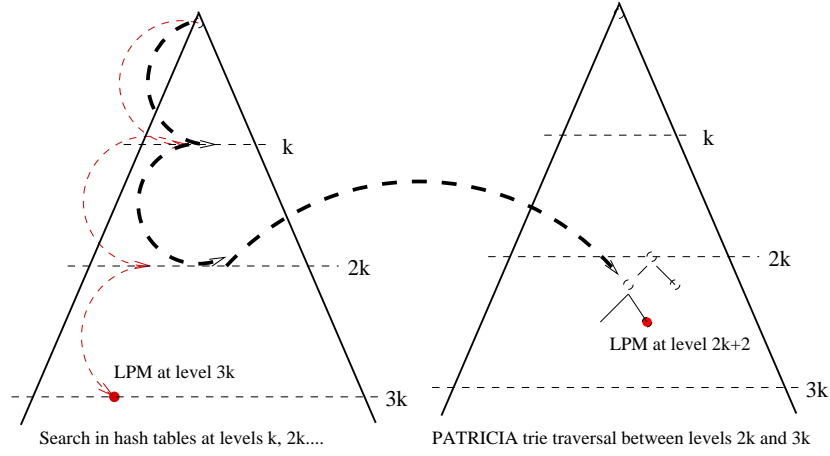


Fig. 2. Multi-level hash table lookup using Elevator-Stairs algorithm. Thick dashed lines depict the lookup hops for LPM at level  $2k + 2$  and thin dashed lines show the lookup hops for LPM at level  $3k$

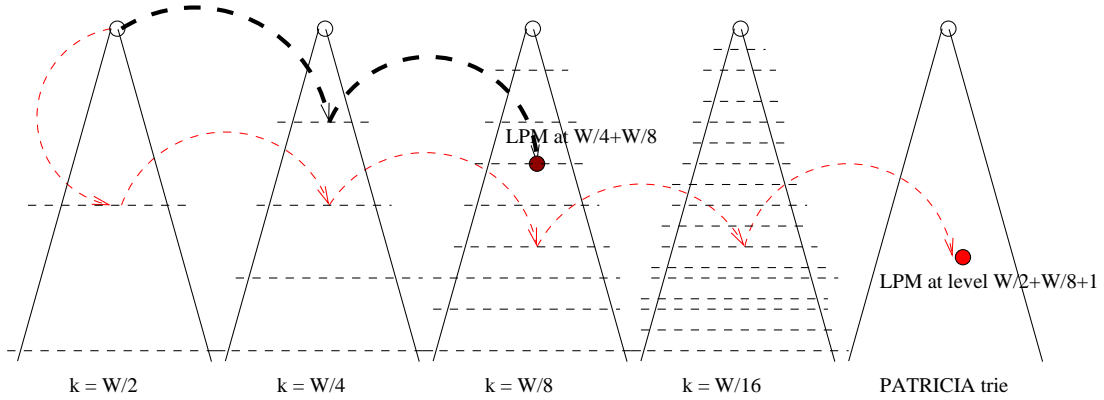


Fig. 3. Multi-level hash table lookup using logW-Elevators algorithm. Thick dashed lines depict the lookup hops for LPM at level  $(\frac{W}{4} + \frac{W}{8})$  and thin dashed lines show the lookup hops for LPM at level  $(\frac{W}{4} + \frac{W}{8} + 1)$

level  $\frac{W}{2}$  by examining if  $(\frac{W}{2} + 1)^{th}$  bit is a match in the PATRICIA tree:

- If the node at level  $\frac{W}{2}$  does not have an edge starting with  $(\frac{W}{2} + 1)^{th}$  bit of  $p$ , the longest match in PATRICIA trie is found, or
- If the node at level  $\frac{W}{2}$  has an edge starting with  $(\frac{W}{2} + 1)^{th}$  bit of  $p$ , search should finish between level  $(\frac{W}{2} + 1)$  and level  $N$ , or
- If there is no hash table entry with  $p[1 \dots \frac{W}{2}]$  as key, the search must finish between levels 1 and  $\frac{W}{2}$ .

In cases (b) and (c), where the longest match in PATRICIA trie is not found, the range of the string depth that the search has to be performed is halved, and the node from which the search has to continue is known. Thus the search range keeps halving by using  $\frac{W}{4}$ -level-tree,  $\frac{W}{8}$ -level-tree,  $\frac{W}{16}$ -level-tree, ..., 2-level-tree and eventually

ends by searching the PATRICIA trie for at most one level unless the longest match in PATRICIA trie is found by case 2 in some tree along the way. All the nodes in the PATRICIA trie store a port number of the longest matching prefix in this algorithm, and the port number can be read from the PATRICIA trie directly. This strategy is chosen to facilitate an update time of  $\mathcal{O}(N)$ . Note that about half of the levels in a  $k^{th}$ -level-tree are never used. For example, a node in level 8 in  $8^{th}$ -level-tree has pointers to nodes in level 16. But these pointers are never used since  $16^{th}$ -level-tree must be used to jump to level 16. Hence, those levels that carry pointers to jump to a level that are already covered by another  $k^{th}$ -level-tree with larger  $k$  can be discarded to save memory.

Figure 3 shows two cases of LPM search, one at level  $\frac{W}{4} + \frac{W}{8}$  (12-bit prefix for IPv4) and the other at level  $\frac{W}{2} + \frac{W}{8} + 1$  (21-bit prefix for IPv4) from the root node.

In the former case, the search fails to find a match in the hash table at level  $\frac{W}{2}$  and subsequently finds successive matches in hash tables at levels  $\frac{W}{4}$  and  $\frac{W}{8}$ . At each of these hash table hits, a single-bit check is made for the existence of further traversals. At level  $\frac{W}{8}$ , it fails the check and thus it is concluded that the LPM for the IP address is at level  $\frac{W}{8}$  and subsequently the NHP information is retrieved from the corresponding node in the hash table. In the latter case, the search consecutively finds a match in hash table at level  $\frac{W}{2}$ , fails to find in level  $\frac{W}{4}$ , finds in level  $\frac{W}{8}$ , fails to find in level  $\frac{W}{16}$ , and finally jumps to the PATRICIA trie to find a match at  $\frac{W}{2} + \frac{W}{8} + 1$  level.

3) *Complexity Analysis:* The best known algorithm for IP routing is the *binary search on prefix lengths* with the complexity of the lookup operation being logarithmic in the prefix length, independent of the number of prefixes. However, updates are complicated ( $\mathcal{O}(N \log W)$ ) due to the use of markers, since for an insertion or deletion of a prefix the precomputed BMP may change for many of the markers that are longer than the new (or deleted) prefix. On the other hand, our *logW-Elevators* algorithm with the same lookup complexity of  $\mathcal{O}(\log W)$  supports fast updating of the data structure with a worst-case complexity  $\mathcal{O}(N)$ . From the analysis of two adjacent snapshots of MAE-West routing table, we observe that 97.34% of the time update occurs at leaf nodes of PATRICIA trie. In that scenario, our *logW-Elevators* algorithm can update in  $\mathcal{O}(\log W)$  time. The *Rope search* algorithm gives same complexities for lookup and update operations. Nevertheless, to achieve that the algorithm requires a memory of  $\mathcal{O}(NW^3)$  outside the router for  $\mathcal{O}(NW^3)$  construction and fast updates. The *Elevator-stairs* algorithm achieves fast lookups comparable to the  $k$ -stride multi-bit trie and the Level-compressed trie schemes, but with optimal memory consumption ( $\mathcal{O}(N)$ ). The  $k$ -stride multi-bit trie scheme implements leaf pushing and expansion of a node to  $2^k$  nodes and hence consumes more memory ( $\mathcal{O}(\frac{2^k NW}{k})$ ). On the other hand, the *Elevator-stairs* algorithm does not perform leaf pushing, and thus consumes optimal memory. Besides, most of the updates occur at leaf nodes of the PATRICIA trie, and the *Elevator-stairs* algorithm can update in  $\mathcal{O}(\frac{W}{k} + k)$  time in such cases. The scalability of the algorithms for IPv6 routing, besides the updating mechanisms, are described in detail in [3].

## V. SHADOW ROUTE CACHING

It becomes increasingly important to add to the network an ability to monitor and verify the validity of routes being used to forward the IP packets, with the rise of intentional

attacks over recent years. However, the problem of detecting invalid routing announcements is a difficult one. The Internet is a large scale, loosely coupled system, without any centralized control or centralized database that provides the latest connectivity information; the only coordination among all the autonomous systems is through running the same standard routing protocol. Although secure communication techniques, such as authentication and encryption, can be used to secure the routing information exchanges as suggested by some recent work [23], not only their wide deployment will take time but also such techniques alone will not assure the ultimate routing reliability due to potential possibilities of routers being compromised. Hence, one of the solutions is to provide a localized monitoring system, at each router, that monitors the traffic pattern and detects any external attacks.

BGP neighbors exchange full routing information when the TCP connection between neighbors is first established. When changes to the routing table are detected, the BGP routers send to their neighbors only those routes that have changed. It is observed that the update of the routing table mainly involves the change of next hop port for a prefix. An external attacker may force the BGP to receive a false NHP information for a prefix. This would result in the forwarding of packets to a wrong node. The proposed scheme is motivated from the above observation that most of the updates in a routing table are change in NHP for a prefix. Whenever a prefix is updated with a new NHP, the prefix with its earlier NHP is cached in the shadow routing table. Thus the shadow routing table maintains a set of prefixes that have been updated recently.

The load at the node is continuously monitored for any traffic surges. When a traffic surge is detected, one of the possibilities is that the forwarded packets are made to return to the node again by making the packets routed in a loop of nodes. This can be done by an intentional attack from external sources. Thus, when the traffic surge is detected, the router retrieves the route prefix recently stored in shadow route cache and uses for packet forwarding. A mechanism is required to be in place for monitoring the surge in traffic and the subsequent behavior once the pre-updated route is retrieved. An entry from the shadow cache is dropped after a suitable interval of time when no surge in traffic is detected. Other mechanisms can be used to detect the looping of packets. For example, packets forwarded at the node are tagged with a code unique for the node. When the network processor (NP) receives packets, it checks for the tag before forwarding and thus detects if the packets are in a loop of nodes. However, this mechanism would cause a slight overhead in the network processor and some

trade-offs between the overhead and the amount of security provided can be studied.

When a packet gets dropped in the network, initially the existing mechanisms are initiated for resending of packets from source to destination. When such mechanisms too fail in delivering the packets, it can be checked that one of the causes would be forwarding of packets to false routes. This can also occur due to some external intentional attack. In that case, the source can send a request to all the nodes on the path to retrieve the pre-updated routes of recent entries in shadow routing cache at each node. An extensive study of this mechanism of detecting and correcting the system, to prevent dropping of packets, is further required.

## VI. SUMMARY AND FUTURE WORK

With advancements in the communication link technologies and the rapidly growing Internet traffic, IP address lookup is becoming a major bottleneck in router performance. The current day routers are expected to perform longest prefix matching algorithm to route millions of packets each second, and this demand on router is increasing even while the prefix search database is expanding in both the dimensions, i.e., IP address length (128 bits for IPv6) and number of prefixes. We presented a BDD based hardware IP address lookup engine, and a software based two algorithms - *Elevator-Stairs* algorithm and *logW-Elevators* algorithm in an effort directed at simultaneously optimizing multiple metrics and provide solutions that scale to IPv6. In this paper, we introduce *shadow route caching*, a mechanism to detect and avoid forwarding packets to bogus routes. Our scheme enables the fast address lookup mechanisms to perform intelligent packet forwarding to provide a trusted Internet routing environment. A detailed study to evaluate and analyze the proposed mechanisms is required.

## REFERENCES

- [1] Xiaoliang Zhao, Dan Pei, Lan Wang, D. Massey, A. Mankin, S. F. Wu, Lixia Zhang, "Detection of invalid routing announcement in the Internet", *Proc. International Conference on Dependable Systems and Networks*, 2002, pp. 59-68.
- [2] Rama Sangireddy and Arun K. Somani, "Binary Decision Diagrams for Efficient Hardware Implementation of Fast IP Routing Lookups", *Proc. IEEE International Conference on Computer Communications and Networks, ICCCN*, October 2001, pp. 12-17.
- [3] Natsuhiko Futamura, Rama Sangireddy, Srinivas Aluru and Arun K. Somani, "Scalable, Memory Efficient, High-Speed Lookup and Update Algorithms for IP Routing", Internal Report, DCNL-CN-2002-001, Department of Electrical and Computer Engineering, Iowa State University.
- [4] *Routing-Analysis, Internet performance measurement and analysis (IPMA) project* [online]. URL: <http://www.merit.edu/ipma>
- [5] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms" *IEEE Network*, Vol. 15, Issue: 2, March-April 2001, pp. 8-23.
- [6] K. Sklower, "A Tree-Based Packet Routing Table for Berkeley Unix" *Proc. Winter Usenix Conference*, 1991, pp. 93-99.
- [7] D. Morrison, "PATRICIA — Practical Algorithm to Retrieve Information Coded in Alphanumeric" *Journal of the ACM*, Vol. 15, No. 4, Oct. 1968, pp. 514-534.
- [8] T. Kijkanjanarat, H. J. Chao, "Fast IP lookups using a two-trie data structure" *Proc. Global Telecommunications Conference, GLOBECOM'99*, Vol. 2, 1999, pp. 1570-1575.
- [9] Henry Hong-Yi Tzeng and Tony Przygienda, "On fast address-lookup algorithms" *IEEE Journal On Selected Areas In Communications*, Vol. 17, No. 6, June 1999, pp. 1067-1082.
- [10] N. Yazdani, P. S. Min, "Fast and scalable schemes for the IP address lookup problem" *Proc. IEEE Conference on High Performance Switching and Routing*, 2000, pp. 83-92.
- [11] M. Waldvogel, G. Varghese, J. Turner, B. Plattner, "Scalable High Speed IP Routing Lookups" *Proc. ACM SIGCOMM '97*, Vol. 27, No. 4, October 1997, pp. 25-36.
- [12] M. Waldvogel, G. Varghese, J. Turner, B. Plattner, "Scalable High-Speed Prefix Matching" *ACM Transactions on Computer Systems*, Vol. 19, No. 4, November 2001, pp. 440-482.
- [13] B. Lampson, V. Srinivasan, and G. Varghese, "IP Lookups using multiway and multicolumn search" *Proc. IEEE INFOCOM'98*, San Francisco, CA, 1998, pp. 1248-1256.
- [14] V. Srinivasan and G. Varghese, "Fast Address Lookups using Controlled Prefix Expansion" *Proc. ACM Sigmetrics'98*, June 1998, pp. 1-11.
- [15] A. Donnelly and T. Deegan, "IP route lookups as string matching" *Proc. 25th Annual IEEE Conference on Local Computer Networks, LCN*, 2000, pp. 589-595.
- [16] S. Nilsson and G. Karlsson, "IP address lookup using LC-Tries", *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 6, June 1999, pp. 1083-1092.
- [17] A. McAuley, P. Tsuchiya, and D. Wilson. "Fast multilevel hierarchical routing table using content-addressable memory." U.S. Patent serial number 034444, 1995.
- [18] P. Gupta, S. Lin and N. McKeown, "Routing lookups in hardware at memory access speeds", *Proc. IEEE INFOCOM'98*, San Francisco, USA, Vol. 3, 1998, pp. 1240-1247.
- [19] Pi-Chung Wang, Chia-Tai Chan, and Yaw-Chung Chen, "A fast IP routing lookup scheme", *Proc. IEEE International Conference on Communications, ICC 2000*, Vol. 2, 2000, pp. 1140-1144.
- [20] T. Chiueh and P. Pradhan, "High-performance IP routing table lookup using CPU caching", *Proc. IEEE INFOCOM'99*, New York, USA, 1999, Vol. 3, pp. 1421-1428.
- [21] D. Pao, C. Liu, A. Wu, L. Yeung, and K. S. Chan, "Efficient Hardware Architecture for Fast IP Address Lookup", *Proc. IEEE INFOCOM'2002*, New York, USA, Vol. 3, 2002.
- [22] D. E. Taylor, J. W. Lockwood, T. S. Sproull, J. S. Turner, and D. B. Parlour, "Scalable IP Lookup for Programmable Routers", *Proc. IEEE INFOCOM'2002*, New York, USA, Vol. 3, 2002.
- [23] G. Huston, "Analyzing the Internet's BGP Routing Table", *The Internet Protocol Journal*, 4(1), March 2001.