2009 IEEE International Advance Computing Conference (IACC 2009)
Patiala, India, 6–7 March 2009

High-performance Longest Prefix Match Logic Supporting Fast Updates for IP Forwarding Devices

Arun Kumar S P
High-End Systems BU
Juniper Networks India Private Limited
Bangalore, India
arunkumarsp@ieee.org

***Abstract*—Lookup architectures are among the well researched subjects in networking. This is due to its fundamental role in the performance of Internet routers. Internet routers use a lookup method known as Longest Prefix Match (LPM) algorithm to determine the next-hop to forward the packet to. State-of-the-art lookup designs try to achieve better search times and/or reduce storage requirements thereby sacrificing the requirement for high update rates. But recent studies have shown the requirement for high update rates, especially in the Internet core routers, due to increasing routing instabilities and anomalous traffic. This paper presents a novel architecture to obtain high update rates in forwarding devices without compromising on the speed and space advantages.**

Keywords **Lookup Table, Longest Prefix Matching**

I. INTRODUCTION

A. Motivation

Internet is considered as one of the great scientific and technological successes of the century. Since inception, the Internet has seen exponential growth in size, topology and the link speeds. This has caused severe strain on the Internet backbone infrastructure in terms of processing speed and link bandwidths requirements. With increasing link speeds and traffic rates, the data plane operations need to be performed efficiently as they work on per-packet basis. One of the critical packet processing tasks performed in data plane of a router is Internet Protocol (IP) lookup or Longest Prefix Matching (LPM). Considering the fact that the LPM is performed on every packet that passes through a router, it is arguably the most run algorithmic problem in the world [1]. Although it had received significant attention in literature over the past decade, much of the work, have focused on two main domains: (a) increasing the speed of the lookup, and (b) decreasing the storage requirements of lookup table. These requirements were satisfied at the cost of the time it takes to make an update in the structure. Recent studies [2], [3], [4] have highlighted the requirement for supporting fast update rates at the core of the Internet. It is also important to note the fact that routing table update operations are generally performed in an atomic manner. i.e. during the update process, lookups are prevented to maintain the consistency of the result. This can lead to increased lookup times and hence degradation in data path performance during routing table updates. Internet core routers typically exchange three to six million updates per day [2] and these have a bursty characteristic which translates to high update rates.

B. Main Contributions and Paper Organization

This paper proposes a novel architecture for performing LPM using associative memory architecture. The proposed scheme eliminates the need for pre-computation on the IP prefixes before populating the routing table. Sorting and partitioning are the main techniques currently employed in IP forwarding devices that use associative memories like Ternary Content Addressable Memories (TCAM's). The proposed architecture substitutes the priority encoder logic of TCAM's with Longest Prefix Finder logic which allows faster updates to the table without compromising the lookup speed advantage of the TCAM based methods.

The rest of the paper is organized as follows: Section II discusses the LPM problem and also takes a glance at prior work and the requirement for high update rates for high performance routers. Section III discusses the proposed architecture and its implementation. Section IV provides a qualitative evaluation of the proposed architecture. Section V concludes the work with suggestion for further research.

II. BACKGROUND

Internet routers use an algorithm known as LPM to select the next-hop information for a particular destination IP address from the routing table. LPM allows routes for large networks to be overridden by more specific host or network routes.

A. Longest Prefix Match

LPM lookups came into existence because of the way in which IP addresses are maintained in the Internet routers. The Internet began with a simple hierarchal addressing scheme in which 32-bit addresses were divided into network address part and host address part. The initial allocation of the address space into four classes- Class A, Class B, Class C and Class D, led to the exhaustion of Class B addresses. This resulted in Classless Internet Domain Routing (CIDR) scheme [5] which assigns

978-1-4244-1888-6/08/$25.00 © 2008 IEEE

multiple contiguous Class C addresses that can be aggregated by a common prefix. With this scheme, network addresses can be allocated as *a.b.c.d/p* where */p* indicates how many bits of the address prefix should be considered for the network address part. Since p can be any arbitrary number of prefix bits between 0 and 32, the allocation becomes more flexible. Although this scheme helped conserving the IP address space, it introduced the complexity of the requirement for the longest matching prefix.

A router maintains a routing table containing millions of entries, each in {address/prefix} form. Associated with each entry is the next-hop information. When the router searches for the prefix matching the destination address present in the packet, it is possible for a packet's destination address to match multiple entries in the routing table. The more the matching prefix bits for a particular entry, the more specific the network address is. Thus the entry with the longest matching prefix will give the better route.

Formally LPM problem is equivalent to *predecessor search*. It is a type of *membership search*, which determines whether a value x is in the set and optionally return some associated data (next-hop) if it is present. *Predecessor search* returns the predecessor *pred*(x,S) of $x \in S$, i.e. the largest element of S that is less than x.

Definition 1: Longest Prefix Match

Consider a universe $U = \{1 \ldots 2^{32}\}$, and a set $T \subset U$ with cardinality n.

$$\forall x \in U; LPM(x,T) = \max\{y \in T \mid y < x\} \quad (1)$$

Figure 1 show how a typical TCAM is used for LPM search. The entries consist of the IP address along with the prefix value. The IP destination address extracted from the packet header is used as the key for the search. As the figure shows, a search for 192.20.11.3 gives three match results. Since the entries are sorted by prefix length the priority encoder gives the address of the longest prefix matching entry. This address can then be used as an index for an SRAM lookup to fetch the next-hop information.

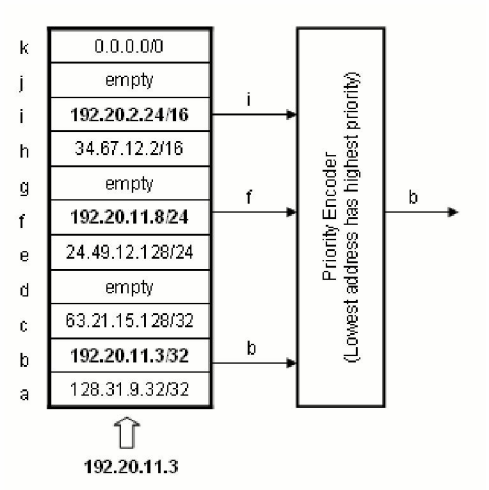

Figure 1. LPM lookup example

B. Related Work

LPM has received significant attention from the academia and industry over the past decades due to the fundamental role it plays in the performance of Internet routers. Most of the current solutions can be broadly classified into two different approaches: algorithmic [6][7][8][9] and architectural [10] [11] [12] .

One of the widely used algorithmic methods for LPM is the binary trie data structure [13]. Optimization of a binary trie by compressing the common sub-sequences is treated in [14] as Patricia Tries. A modified version of this algorithm [15] was incorporated in the BSD networking stack. Multi-bit tries with compressed nodes was used in Lulea [6] and Tree Bitmap [16] methods. Although Lulea scheme provides a worst case memory access count of 12 and a considerable reduction is storage size, the update rate suffers due to the use of leaf pushing. The Tree Bitmap method provides a better update rate than Lulea by avoiding leaf pushing. But this solution is not scalable. Binary Search on Prefix Lengths method described in [8] provided an efficient way of performing lookup with bounded memory accesses but fails due to the pre-computation involved in creating the database. Various classification methods like tuple space, HiCuts, and grid-of-tries can also be applied to this domain as prefix matching is a single dimension version of these algorithms.

Architectural solutions are predominantly based on Content Addressable Memories (CAM). Given an input key, the CAM compares the key with all the stored elements in parallel, which enables it to produce the output by consuming very few clock cycles. This makes it an ideal choice for exact match lookups. But the Internet address space is not strictly hierarchical and the search entries involve arbitrary prefix lengths. Although development of Ternary CAMs enabled arbitrary prefix lookups by storing an additional "Don't Care" condition, this lead to the segmentation of the space. The priority encoder based architecture of the CAMs made pre-computation of the prefixes mandatory before any update or addition to the table.

C. Relevance

Analysis of current solutions used for IP Lookup reveals that high update rates were not one of the key design criteria used for the development of these algorithms. Often solutions find a satisfactory compromise between the lookup time budget and the table storage space requirements. But recent studies [17] projecting the requirements of future IP routers show that the load of running routing protocols like Border Gateway Protocol [18] is not confined to storing and processing large routing tables. BGP-speaking routers must be able to handle second-by-second incremental fluctuations in the routing tables. By observing the dynamic BGP activity during 2005 inside AS1221, the authors of [17] have found that the number of update messages have almost doubled during 2005, growing from ≈260,000 per day to ≈550,000 per day. It is also interesting to highlight the fact that the update rate has doubled while the size of the routing table grew only by 18% during 2005. Although before mentioned numbers may translate to low average updates per second ratios, it is evident from Figure 2 that updates happen in bursts which can translate to high

transient update rates. It can be argued that deployment of route dampening algorithms [19] and the increased use of route aggregation [20] will significantly reduce this problem of Internet routing instability and thereby the updates. But recent statistics collected from various autonomous systems statistics show otherwise. Figure 2 show the update rate statistics obtained from AS65000 for the time period from 2001 to 2008. It is evident that there is an increased update activity during 2007-2008. This can be attributed to the increasing number of redundant or multi-homed connections to the Internet via multiple service providers. It is being widely known that multi-homing can break the aggregate address blocks. Also the prevalence of multi-homing exhibits a relatively linear growth rate [21].

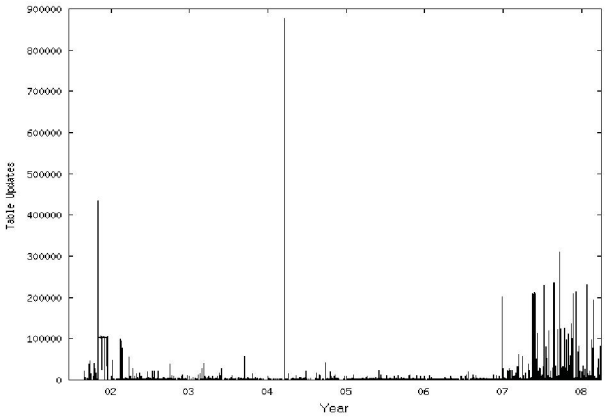

Figure 2. BGP Prefix updates. Data collected from AS65000

An architecture that can perform updates at line rates can guarantee high performance packet forwarding as against statistical guarantees. This also can make the Internet infrastructure robust against temporal bursty traffic characteristics generated due to malicious activities. In [22], authors noted that a surge in BGP updates coincided with events such as the outbreak of SQL Slammer worm [23]. Although the attack was not directly targeted at the Internet infrastructure, a number of AS-AS peering links were operating at critical load thresholds during the attack period. Similar dynamics were observed during the Code-Red [24] and NIMDA [25] worm attacks. Usage of architectures that support high update rate without compromising on lookup performance can harden the Internet infrastructure against such events.

III. ARCHITECTURE

The proposed architecture consists of Prefix Entry Units (PEU's) and Longest Prefix Finder (LPF) logic (Figure 3). Each PEU's consist of storage elements for the IP address value and the prefix along with the elements for performing the comparison operations. The PEU's take the *key* as external input and compares the relevant part of the stored address value with the *key*. A match found (mf_i) is asserted in case of a match. The prefix value along with the 'match found' signal is produced to the LPF logic for LPM determination, The LPF logic compares the prefixes whose associated match found signal is asserted and provides the address of the longest prefix entry. Associated next-hop values can be stored in the LPF or optionally encoded address can be provided for an external lookup for next-hop.

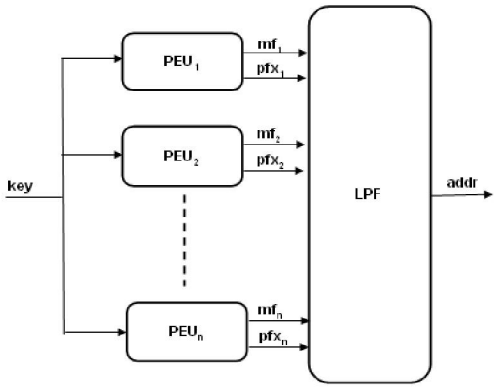

Figure 3. Block diagram

A. Prefix Entry Design Unit

Each Prefix Entry Unit (PEU) stores an address value ($addr_i$) along with its associated prefix or mask (pfx_i). The input to the unit includes a *key* and the outputs consist of prefix (pfx_i) and the match found flag (mf_i). The *key* is generally extracted from the packet header by parser logic present in the data-path. For IPv4 packets it is the 32-bit destination IP address.

The functionality of the PEU's can be formulated as

$$\forall i \in \{1..n\} \quad mf_i = (key \bullet pfx_i) \Leftrightarrow (key \bullet addr_i) \tag{2}$$

The logic for PEU can be directly derived from the above Boolean Equation (2) and is shown in Figure 4.

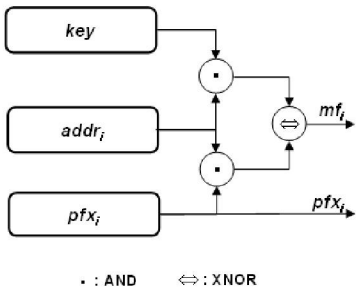

Figure 4. PEU implementation

The prefix values (pfx_i's) are also produced to the LPF for comparison. The each of the match found flag (mf_i) serves as an enable or valid signal for the corresponding prefix outputs (*pfxi*). The mf_i is asserted only when the condition specified in Equation (2) is satisfied. This enables the LPF logic to do a comparison of only prefixes that produce a match signal.

B. Longest Prefix Finder Design

The Longest Prefix Finder performs the comparison of the prefixes in a novel way and finds the longest of the prefixes produced to it. The prefixes are flagged by a match found

signal which indicates whether the associated prefix belongs to a matched entry or not. Comparison of the prefixes is performed only on matched prefixes (i.e. only on pfx_i's who's $mf_i = 1$). This guarantees that the prefix obtained is a Longest Prefix Match entry.

Comparisons are performed on the matched prefixes in a parallel bit-wise manner. A bit slice form all the matched prefix entries take part in the operation. The operation is performed from the Least Significant Bit (LSB) towards the Most Significant Bit (MSB). The architecture exploits the manner in which prefixes are encoded. Prefixes have consecutive 1's followed by 0's. This enables the operation to be completed when it hits logic 1 in any of the bit slices it is comparing. And that particular entry is the longest matched prefix and its address can be used to fetch the associated value (next-hop).

C. Methodology

Consider the following prefix set:

$\{p_{i,j}\}$; $i \in \{1 \ldots n\}$ and $j \in \{1..32\}$, *where n is the number of prefixes*

By this notation, $p_{1,j}$; $j \in \{1...32\}$ will represent the 32 bits of the first prefix. Also $P_{1,0}$ is the LSB of the first prefix and $P_{n,32}$ is the MSB of the nth prefix. The algorithm starts with the bit slice $p_{i,1}$ i.e. LSB's of the prefixes. Only enabled prefix bits take part in the operation. If all bits in the slice are logic 0's then the comparison shifts to next prefix bit slice. If any of the bits in the participating slice is logic 1, a match flagged and the operation ends. Also all the bits of $p_{i,j}$ for a fixed value of i and $j \in \{1..32\}$ is refereed to as a bit-slice. The algorithm proceeds from the bit-slice constituting the LSB's towards the bit-slice representing the MSB's. Each prefix bit has a Prefix Bit Logic Block (PBLB) associated with it. A PBLB is active only if its enable signal is asserted. The PBLB's of the LSB's are mapped to their corresponding mf_i's. All active PBLB's for a bit-slice concurrently check if associated prefix bit is logic 1, in which case it asserts a match indicator *mi* and stops the operation. If the prefix bit logic 0 and if there are no other bits in the same slice with logic high, then it advances to the next bit slice. Since all this can be implemented as a combinational circuit, the complete operation can be performed within the time required for a typical external memory access.

Unlike in CAM based architectures, this approach doesn't require modifications to the table ordering before or after an updation. Updation can be performed at random locations in *O(1)* time. A stack can be maintained by the software for identifying the free locations. Insertions can be done by writing the value to the address provided by a stack pop which is a *O(1)* operation. Deletions will require the address of the deleted entry to be pushed into the stack.

D. Implementation

D Each of the Prefix Bit Logic Block ($PBLB_{i,j}$) has an enable signal *en(i,j)* which it receives it from the block left to it. The enable signal to the LSB of each prefix ($PBLB_{i,j}$) is mapped to the mf_i obtained from the PEU_i. PBLB's also take a feedback input *f(i)* which indicates whether any of the PBLB's in its bit-slice is a logic 1. The logic block result signal *lbr(i,j)* is asserted if data *d(i,j)* is high for an enabled block. This is used to generate the feedback input and the match indicator *mi(i,j)*. The enable signal for the next bit *en(i,j+1)* (and thereby all the remaining bits) in an entry is flagged only if the current PBLB is enabled and the feedback indicates that all the bits in the bit-slice is a logic 0. This when translated to hardware logic will result in the implementation shown in Figure 5.

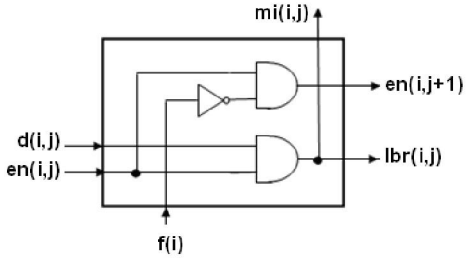

Figure 5. PBLB implementation

Figure 6 shows the high level architecture of the LPF using the above mentioned PBLB logic.

IV. Evaluation

The proposed architecture has *O(1)* lookup and update times. Updation will require another stack read/write depending on whether the operation is insertion/deletion which are also *O(1)* operations. Worst case time for lookup is dependent on the combinational delay introduced by the LPF logic which is bounded by the 32 bit slices for IPv4. But in practical scenarios the results will be better because of the nature of the prefix encoding and the manner in which search is performed. Figure 7 shows the average prefix length observed in a core router from AS 65000. This indicates that the LPF circuit is highly likely to get a hit by the time it traverses 10 bit-slices.

Table I compares the lookup performance and update overhead of various state-of- the art schemes with the proposed architecture. Tuple space [26] is a classification algorithm has a worst case latency complexity of W^{d-1}, where d is the number of fields and W is the average width. For a prefix lookup is a single dimensional classification with a width of 32, which makes it *O(1)*. Similar is the case with HiCuts [27] and grid-of-tries [28] which have a lookup complexity of *dW*. But all of these have an overhead of pre-computation which degrades the update performance. Pipelining support which helps in increasing the update performance is not supported in tuple space, HiCuts and grid-of-tries.

TABLE I. Performance Comparison

Method	Precomputation for updates	Pipelining support
TCAM	yes	yes
tuple space [28]	yes	no
HiCuts [29]	yes	no
grid-of-tries [30]	yes	no
Proposed method	no	yes

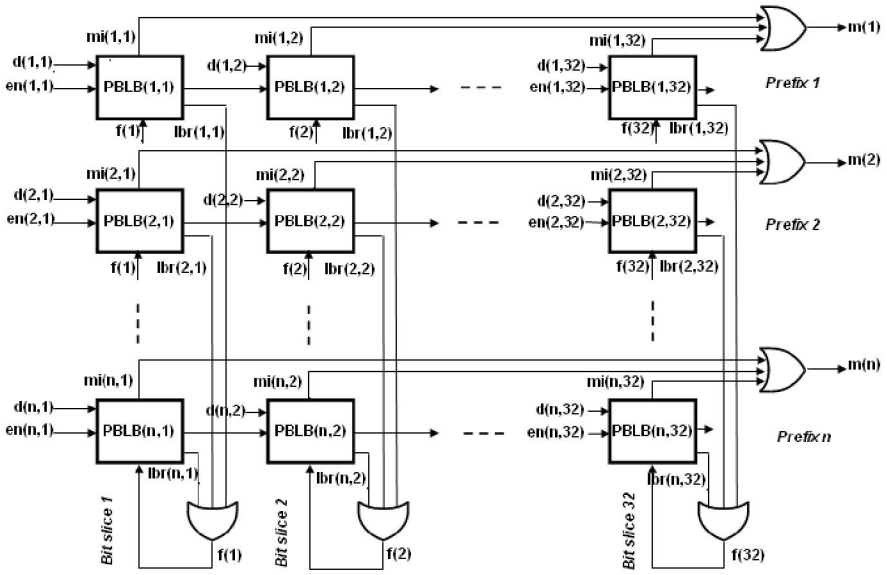

Figure 6. PLF Implementation

Table II compares the storage requirements for the various schemes discussed. It reveals that the proposed architecture has similar storage requirements as that of a TCAM but considerably less than the other methods. Thus it provides the advantages of a TCAM architecture (*O(1)* lookup complexity and lower memory size) eliminating the requirement of pre-computation for updates.

TABLE II. STORAGE REQUIREMENTS FOR 1500 ENTRIES

Method	Storage size (approx.)
TCAM	20kB
tuple space [28]	350kB
HiCuts [29]	70kB
grid-of-tries [30]	700 kB
Proposed method	20kB

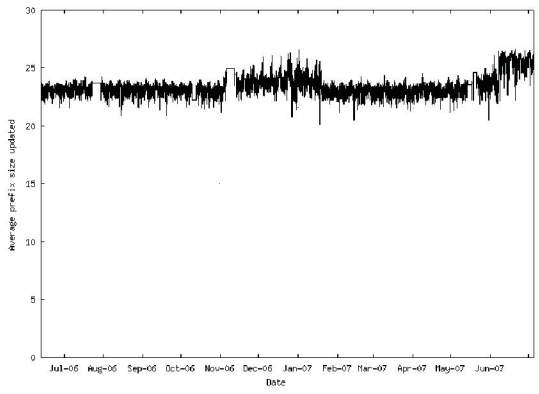

Figure 7. Average prefix size

V. CONCLUSION AND FUTURE WORK

This paper proposes a method for designing packet processing engines for IP lookup which support high lookup and high update rates. By employing a high-speed longest prefix finding logic it removes the pre-computation or partitioning that is required in current implementations for an update to the table. This enables wire speed updation to the table without compromising the lookup rate.

Although the architecture doesn't consume more space that the present TCAM based architectures, optimization of space is an open issue for further research. This is also an important requirement considering the rate at which routing table entries are increasing at the Internet core.

REFERENCES

[1] Mihai Patrascu: "Searching the Integers"

[2] Craig Labovitz, G.Robert Malan, and Farnam Jahanian, "Internet Routing Instability," ACM/IEEE Trans. on Networking, vol. 6, no. 5, pp. 515-528, Oct. 1998.

[3] Jun Li, Michael Guidero, Zhen Wu, Eric Purpus, Toby Ehrenkranz, " BGP routing dynamics revisited", Computer Communication Review 37(2): 5-16, 2007

[4] Mohit Lad, Xiaoliang Zhao, Beichuan Zhang, Dan Massey, and Lixia Zhang , "An analysis of BGP update burst during Slammer worm attack". In Proceedings of 5th International Workshop on Distributed Computing, December 2003.

[5] Fuller, V., Li, T., Yu, J., and K. Varadhan, "Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy" RFC 4632, Internet Engineering Task Force, August 2006.

[6] M. Degermark,A. Brodnik,S. Carlsson,and S. Pink "Small forwarding tables for fast routing lookups". In ACM SIGCOMM, 1997.

[7] B. Lampson, V. Srinivasan, and G. Varghese. "IP Lookups using Multiway and Multicolumn Search". In Proceedings IEEE Infocom, 1998.

[8] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. "Scalable High-Speed IP Routing Lookups". In ACM SIGCOMM, 1997.

[9] V. Srinivasan and G. Varghese. "Fast IP Lookups Using Controlled Prefix Expansion". In ACM SIGMETRICS, 1998.

[10] P. Gupta, S. Lin, and N. McKeown. "Routing Lookups in Hardware at Memory Access Speeds". In Proceedings IEEE INFOCOMM, 1998.

[11] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor. "Longest Prefix Matching using Bloom Filters". In ACM SIGCOMM, August 2003.

[12] M. J. Akhbarizadeh, M. Nourani, R. Panigrahy, S. Sharma. "A TCAM-Based Parallel Architecture for High-Speed Packet Forwarding". IEEE Trans. Computers 56(1): 58-72, 2007.

[13] Donald Knuth, The Art of Computer Programming, Vol.3 — Sorting and Searching. Addison-Wesley.

[14] D R Morrison. "PATRICIA - Practical Algorithm To Retrieve information Coded In Alphanumeric". Journal of ACM 15, 4 (Oct.), 514–534.

[15] Keith Sklower. "A Tree-based Packet Routing Table for Berkeley Unix." In USENIX Winter 1991. 93–104.

[16] W. N. Eatherton. "Hardware-Based Internet Protocol Prefix Lookups". Thesis, Washington University in St.Louis, 1998.

[17] G. Huston and G. Armitage. "Projecting Future IPv4 Router Requirements from Trends in Dynamic BGP Behaviour". In Australian Telecommunication Networks and Applications Conference (ATNAC), 2006.

[18] Y. Rekhter, T.Li, S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Internet Engineering Task Force, January 2006.

[19] Curtis, V., Chandra, R. and R. Govindan, "BGP Route Flap Damping", RFC 2439, November 1998.

[20] Y. Rekhter and C. Topolcic, "`Exchanging Routing Information Across Provider Boundaries in the CIDR Environment," RFC 1520. September 1993.

[21] R. Govindan , A. Reddy, "An Analysis of Internet Inter-Domain Topology and Route Stability", Proceedings of the IEEE INFOCOM '97.

[22] Mohit Lad, Xiaoliang Zhao, Beichuan Zhang, Dan Massey, and Lixia Zhang, "Analysis of BGP Update Surge During Slammer Worm Attack," in Proc. of 6th International Workshop on Distributed Computing (IWDC), December 2004.

[23] CERT Advisory CA-2003-04, "SQL Slammer"

[24] CERT Advisory CA-2001-19, "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL,"

[25] CERT Advisory CA-2001-26, "Nimda Worm"

[26] V. Srinivasan, "A packet classification and filter management system," in Proc. IEEE INFOCOM, vol 3, Apr. 2001, pp 1464-1473.

[27] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in Proc. Hot Interconnects 7, Stanford University, Stanford, CA, Aug 1999.

[28] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," Computer Communication Review, vol 28, no. 4, pp. 191-202, Oct. 1998.