Range Trees with Variable Length Comparisons

Ioannis Sourdis, Ruben de Smet, and Georgi N. Gaydadjiev
Computer Engineering
EEMCS, TU Delft, The Netherlands
{sourdis, ruben, georgi}@ce.et.tudelft.nl

***Abstract*— In this paper we introduce a new data structure for address lookup, a new tree structure which improves on the existing Range Trees allowing shorter comparisons than the address width. The proposed scheme shares among multiple concurrent comparisons common address prefixes and suffixes and also omits address parts not required for computing a next node branch. In so doing, for a given memory bandwidth, we achieve a larger number of concurrent comparisons than the original Range Tree. This results in less memory accesses and lower latency per lookup. Performance scales better as the address width and the number of address ranges increase. We describe the rules employed to construct the proposed structure and offer two heuristics which generate the "configuration" of the decision tree given a set of address ranges. The proposed Range Tree with variable-length comparisons (RT-VLC) has up to 50% less tree-levels than the original Range Tree and its memory requirements are 50% to 2× that of a linear search approach.**

***Index Terms*— Address Lookup, IP Lookup, Range Tree.**

I. Introduction

Address lookup is an essential function that finds application in several domains: primarily in IP lookup [1] and packet classification [2, 3], but also in interprocessor communication -at the newly proposed progressive address translation [4]. Internet backbone routers use packet's destination address to determine the next hop of a packet, they contain hundreds of thousands entries in their tables and require to perform millions of lookups per second. The rapid growth of internet traffic and the growing size of routing-tables make more difficult to keep pace with the increasing need for faster processing rates. IPv6 growth increased 300% in the past two years and coupled with the IPv4 exhaustion poses the need for solutions scalable with the address width [5].

Although a plethora of algorithms have been proposed, several of the above challenges remain open. On one hand, various Trie-based data structures may provide fast updates but they are either unbalanced or require high memory resources. Their latency does not scale well with the address width - especially when they rely on a long first-stride to reduce their height [6]. On the other hand, Range Trees are balanced and hence have lower latency [6], but the number of node branches is limited to the ratio of memory bandwidth to address width. The above leave as the main commercially available solution the power hungry and not scalable TCAMs.

In this paper we introduce a Range Tree with variable-length comparisons (RT-VLC), a new data structure for address lookup, and attempt to address the above challenges. As opposed to the original multiway Range Tree which performs comparisons of complete addresses, in the RT-VLC comparisons of fewer bits suffice to select the correct node branch. More precisely, *address prefixes and suffixes can be omitted* from processing when certain constraints are met, while *common address prefixes and suffixes of concurrent comparisons are shared*. In so doing, we increase the number of comparisons per node for a given memory bandwidth and thus reduce the number of required tree levels, memory accesses, and lookup latency. In addition, the above improve the scalability in terms of lookup latency as the address width and the routing table size increase.

The remainder of the paper is organized as follows: section II offers some background on address lookup algorithms. In section III we describe the RT-VLC and the heuristics employed to generate the configuration of the proposed scheme. Then, in section IV we evaluate the performance, memory requirements and scalability of the RT-VLC. Finally in Section V we draw our conclusions.

II. Background

Various algorithms have been proposed for address lookup. They have been summarized in several surveys such as the ones by Gupta and McKeown [2] and by Taylor [3] which focus on packet classification (lookup in multiple fields), or the one by Ruiz-Sanchez et al. that emphasizes more the algorithmic side of the various approaches [1].

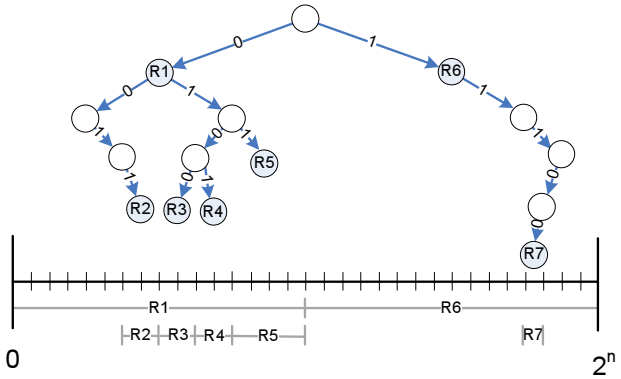
As illustrated in Figure 1(a), the set of different address ranges can be expressed as *intervals*, where the complete bit patterns of the addresses can be compared to perform a lookup, or *prefixes* out of which the longest matching one should be reported (Longest Prefix Match). Ruiz-Sanchez et al. indicate that address lookup involves searching in two dimensions: *length* and *value* [1].

Tries can be considered a "search on length" approach as they perform a sequential search on the length dimension, matching at step n prefixes of length n. Improvements on the basic Trie structure may include binary search on length instead of sequential [7], path compression e.g. PATRICIA [8], and fixed or variable-stride multibit tries e.g. [9]. Figure 1(b) depicts an example of a simple Trie structure. We can observe that the resulted decision tree can be significantly unbalanced; as indicated in [1] it is difficult to control the height of a Trie which does not scale in the address width. Furthermore, the memory requirements of the Tries are relatively high. Multibit tries improve on the tree height but not on the scalability, while they significantly increase their memory consumption.

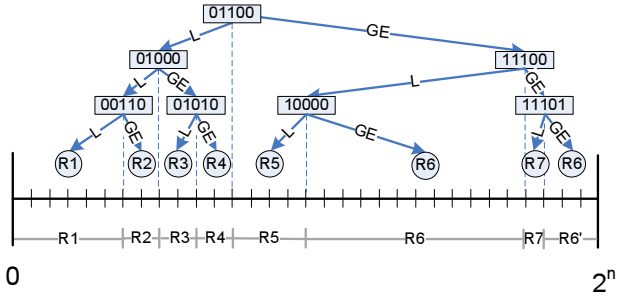
A typical "search on values" approach is the Range Tree, which is depicted in Figure 1(c). Range Trees avoid the

Prefixes	Intervals
R1: 0*	R1: [00000,00110)
R2: 0011*	R2: [00110,01000)
R3: 0100*	R3: [01000,01010)
R4: 0101*	R4: [01010,01100)
R5: 011*	R5: [01100,10000)
R6: 1*	R6: [10000,11100)
R7: 11100	R7: [11100,11101)
	R6': [11101,100000)

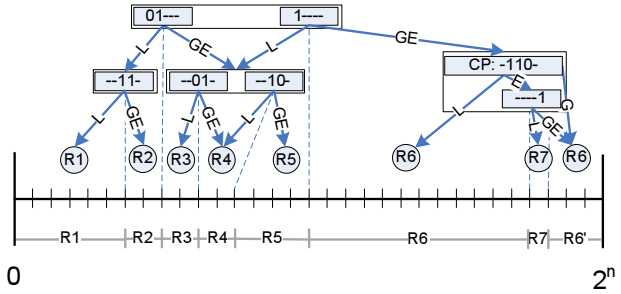
(a) Set of address ranges as prefixes or intervals.



(b) Trie



(c) Range Tree



(d) RT-VLC: Range Tree with Variable Length Comparisons. CP: Common Prefix, '-': bit not necessary to be processed, L: less, E: equal, GE: greater or equal, G: greater.

Fig. 1. Data structures for address lookup: Tries, Range Trees and the proposed Range Tree with variable length comparisons.

length dimension performing comparisons on the expanded prefixes (full addresses). They perform one or many address comparisons at each step creating a balanced decision tree. Range Trees need to store the complete addresses to be compared at each stage and therefore consume considerable memory size. Multiway Range Trees read and compare at every step multiple addresses; this limits their number of ways to the available memory bandwidth and reduces their scalability with respect to the address width.

III. RT-VLC: Range Tree with Variable Length Comparisons

The Range Tree with Variable Length Comparisons (RT-VLC) is a multi-way tree that allows processing of fewer address bits per comparison and therefore, given a memory bandwidth, increases the total number of comparisons performed in a single step. In doing so, the decision tree has more branches per node and consequently its height is reduced vs. the original Range Tree which uses the same memory bandwidth and compares complete addresses.

Range Tree nodes closer to the root compare addresses that are sparser in the address space and therefore their suffixes can be omitted without creating imbalance on the tree. In addition, nodes closer to the leafs compare addresses that are denser in the address space and therefore their prefixes can be either omitted or shared. We capitalize on the above observations to create the RT-VLC. It is expected that a RT-VLC will start comparing address prefixes at the root node omitting the suffixes. Then, gradually at the next levels will continue with the address infixes and will end up comparing suffixes at the leafs. Intuitively, as we traverse the RT-VLC the common address prefixes will gradually get longer and the shared or omitted address suffixes will get shorter. That is because the search space will gradually reduce and better "precision" will be required.

The RT-VLC has the following properties:

- A node maps to an address range of the address space. The union of the children node address ranges is the address range of their parent node.
- Address suffixes can be omitted from processing, when they are *zero*. [1]
- Comparing common address prefixes and suffixes is shared among concurrent comparisons.
- The common prefix of the node borders can be omitted from processing.

Figure 1(d) illustrates a RT-VLC which increases the number of node branches at the decision tree comparing fewer address bits. In this example we assume available memory bandwidth of 5 bits equal to the one of the Range Tree in Figure 1(c). We next discuss the above RT-VLC example, and show how comparing fewer address bits can be sufficient, while address parts can be omitted or shared. At the root node, comparing the two most significant bits "01---" and the most significant bit "1-----" is the equivalent of comparing the complete addresses "01000" and "10000". In the second iteration and after taking the middle root branch, we do not need to compare the two most significant bits since after the first step we know the incoming address is "01xxx". Similarly, after taking the right branch of the root node we know that the most significant bit is "1xxxx". Then, the two addresses to be compared ("11100" and "11101") have a common prefix ("-110x") which is shared and compared separately. The decision of that node is based on the outcome of the common prefix comparison and (if needed) the comparison of the least significant bit. The above example, results in a well balanced

[1]During the construction of a RT-VLC one may force an address suffix to zero, loosing in precision, but reducing the required address bits.

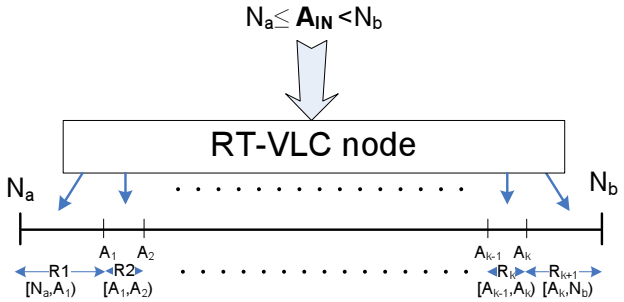


Fig. 2. Generic view of a RT-VLC node *N*.

decision tree which is less deep than the one of the Range Tree using less memory bandwidth.

A. RT-VLC Rules

The RT-VLC exploits four rules to increase the number of branches per node given a specific memory bandwidth in order to reduce the depth of the decision tree. Below we describe these rules. We consider a RT-VLC node N as illustrated in Figure 2 that maps to the address range $[N_a, N_b)$ and divides it in $k+1$ subranges $R_1,\ R_2,\ ...,\ R_{k+1}$ defined by the unique addresses (of width W) $A_i \in [N_a, N_b)$, $\forall i \in \mathbb{N},\ i \leq k$, such that $R_1 = [N_a, A_1)$, .., $R_i = [A_{i-1}, A_i)$, ... $R_{k+1} = [A_k, N_b)$. Then an incoming address $A_{IN} \in [N_a, N_b)$ needs to be compared against the addresses A_i in order to determine the subrange R_i to which it belongs. It is noteworthy that a single comparator reports whether the A_{IN} is "L: Less" or "GE: Greater or Equal" to an address A_i (A common prefix comparator reports also equality). The construction of a RT-VLC is based on the following rules (formal proofs omitted due to lack of space):

Rule 1: Omit the common prefix of the node borders: When there is a common prefix of length L ($L < W$) of the node borders N_a and N_b then the L most significant bits of the addresses A_i can be omitted from the comparisons at the node.

That is because all addresses $A_i, A_{IN} \in [N_a, N_b)$ will have the same common prefix of their L most significant bits. This means that the comparison of the L MSbits between A_{IN} and A_i will always result in equality, enabling the comparison of the $W - L$ LSbits to produce the result.

Rule 2: Omit address suffix of value '0'. Let an address A_i have a suffix of length L, where $L < W$, that is zero. Then, this suffix of A_i does not need to be compared against the L last bits of A_{IN}.

In order for the comparison of the L-bit suffix to affect the final result of the comparison, the $W - L$ prefix comparison would result in equality (which translates into "GE"). In that case however, the comparison between the A_{IN} and A_i L-bit suffix will always be "GE" since the A_i L LSbits are zero. Then, since "GE" is already the result reported by the prefix comparison we can omit the zero suffix comparison.

Rule 3: Share addresses' common prefix. The Common Prefix A_i^{CP} of the addresses A_i of length L ($L < W$) can be shared among concurrent comparisons and processed separately. Then if the A_{IN} prefix of length L is *Less* than A_i^{CP}, then $A_{IN} \in R_1$ (the first address range of the node), if it is "Greater" than A_i^{CP} then $A_{IN} \in R_{k+1}$ (the last address range of the node), else we consider the result of the $W - L$ bits suffix to determine where A_{IN} belongs to.

The comparison of two numbers is performed starting from the MSbit towards the LSbit, then the Common Prefix A_i^{CP} of the addresses A_i can be shared and processed separately. In case of prefix equality between A_i^{CP} and the L MSbits of A_{IN} the suffix comparisons will determine the final result, otherwise the result is determined by the prefix comparison and is either the first R_1 or last R_{k+1} address range when A_{IN} prefix is Less or Greater than A_i^{CP}, respectively.

Rule 4: Share addresses' common suffix. The common suffix A_i^{CS} of the addresses A_i, of length L ($L < W$) can be shared among concurrent comparisons and processed separately. Let $R_p = [A_{p-1}, A_p)$ ($p \in \mathbb{N},\ 1 \leq p \leq k+1$) be the address range that the comparisons of the $W - L$ MSbits of A_i and A_{IN} indicate. Then $A_{IN} \in R_{p-1} = [A_{p-2}, A_{p-1})$ IFF all three statements below are true:

(i) the A_{IN} $W - L$ MSbits are equal to the ones of A_{p-1},
(ii) the A_{IN} suffix of length L is less than A_i^{CS},
(iii) $R_p \neq R_1$

Otherwise $A_{IN} \in R_p$.

In essence, the suffix comparison will change the result from R_p to R_{p-1} only when (i) the A_{IN} prefix is equal to the prefix of the low bound of R_p (A_{p-1}), otherwise the comparison result is already determined by a more significant bit, (ii) the result of the suffix comparison (between A_{IN} and A_i^{CS}) is "Less", otherwise we already have the correct address range R_p, and (iii) the prefix comparison does not output the address range R_1 since in that case $R-p-1 = R_0$ which is not valid.

The above RT-VLC Rules can be applied independently as they do not affect each other. For example, we can omit common node prefix (Rule 1), omit any zero suffix (Rule 3) , then share address common prefix (Rule 2) and suffix (Rule 4) of the remaining address bits, and finally separately compare the remaining bits for each A_i.

B. Constructing a RT-VLC

We propose two heuristics to construct a RT-VLC based on an arbitrary set of address ranges. Both use recursive functions which generate the configuration of a tree node or tree level. We follow two approaches, namely a *top-down* and a *bottom-up*. The top-down heuristic creates the root node first and then similarly moves to its children and towards the leafs of the tree. The bottom-up heuristic constructs the leaf nodes first and subsequently their address bounds are used for the next tree level; this is repeated until the root of the tree is reached. A heuristic should be tailored for a specific implementation and hence may allow comparisons of only few address lengths and only a single length in simultaneous comparisons.

a) **Top-Down Heuristic**:

1) Apply the RT-VLC rules 1 and 2.
2) Select a comparator length that maximizes the possible number of branches, i.e., 8-bits.
3) Consider all addresses in the set to be processed with the above comparison length. Omit address suffixes that cannot be compared (due to comparison length) assuming they are equal to zero (and apply accordingly Rule 2).
4) Based on the above create the defined from the comparisons groups/intervals.
5) If necessary merge neighboring groups until the number of comparisons are reduced to the available comparator resources. Take into account the RT-VLC rules 3 and 4. Merging aims at creating groups which contain a balanced number of address ranges. The resulted groups are the node branches and the groups' borders the comparisons to perform.
6) Recursively repeat for the created children nodes.
7) Terminate when each node branch points to a single address range.

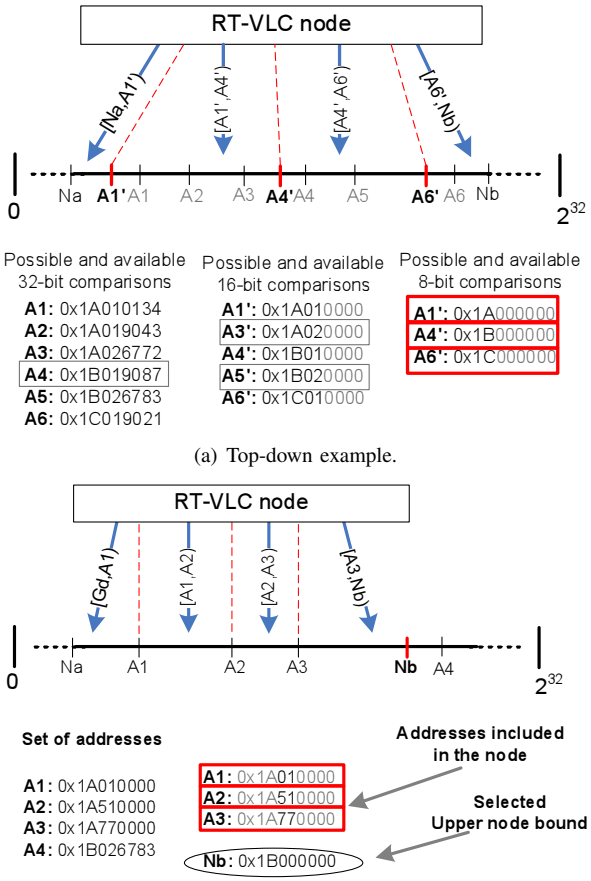
Figure 3(a) illustrates a simple example of choosing address bounds to be included in a node in a top-down heuristic. In this example there are no common address parts to be shared or omitted. Addresses $A_1, ..., A_6$ are available in the node that maps to $[N_a, N_b)$. Let us assume we have 32-bits available for storing addresses to be compared in a node. This means that considering 32-bit comparisons only one address can be included in the node, e.g., A_4. Considering 16-bit comparisons, assuming the 16 LSbits of each address are '0', we may choose 2 out of 5 available addresses to be compared, e.g., A'_3 and A'_5. Finally, for 8-bit comparisons we can perform all three possible address comparisons A'_1, A'_4, and A'_6, which appears to be the most efficient choice.

b) **Bottom-Up Heuristic**:

1) Select the first b addresses $A_i > N_a$ (where N_a initially is 0) that can be compared in one node after applying the RT-VLC rules (e.g., addresses $A_i, A_{i+1}, ...A_b$).
2) Set node's upper bound N_b a point in the address space where $N_b < A_{b+1}$ that N_a has a long suffix of 0's[2]. The resulted node maps to an address range $[N_a, N_b)$.
3) Repeat the above starting from the upper bound of the previous group ($N_a^{new} = N_b^{prev}$) until all addresses A_i in the address space are grouped.
4) Recursively repeat the above steps for the next upper tree level $L-1$ using as new set of addresses A_i^{L-1} the borders N_i^L of the previous level.
5) Terminate when all addresses in the list are processed in a single (root) node.

Figure 3(b) illustrates a simple example of choosing node address bounds in a bottom-up heuristic. Staring from N_a and assuming 32-bits available for storing node address parts, then, addresses A_1, A_2, and A_3 can be included in a single node. The addresses $A_1, ..., A_3$ have a common prefix of one byte and zero suffixes of two bytes, consequently, the node needs to store one byte per address and their common prefix. Subsequently, an upper node bound N_b needs to be chosen. The address with the longest suffix of zero before A_4 is chosen for N_b in order to minimize the required comparison bits at the upper tree-level. In this example, this address is N_b =0x1B000000, which will also be the low node bound of the next node to be constructed at the same level N_a^{new}.

[2]The longer the zero suffix the fewer the bits that need to be processed at the upper level. There may be a threshold set on the minimum number of addresses A_i to be included in the node, e.g. 75%; the rest of the addresses can be given away in exchange for a "better" group bound with longer suffix of zeros.



(a) Top-down example.

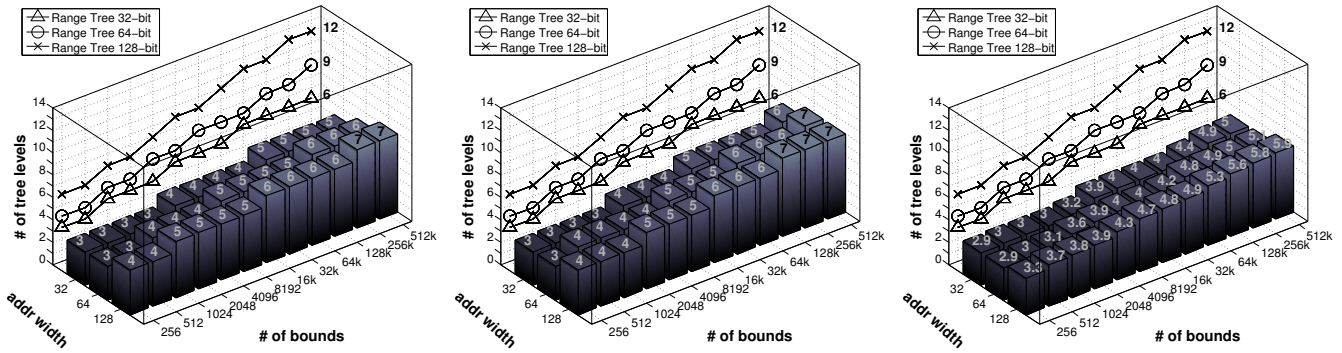
(b) Bottom-up example.

Fig. 3. Examples of selecting addresses to be included in a RT-VLC node.

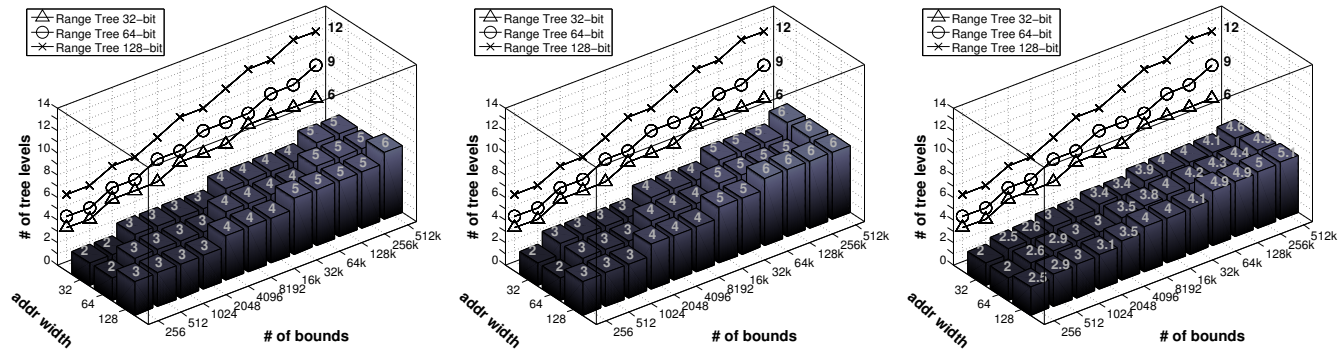
C. Incremental Updates

Most applications using address lookup need to update their set of address ranges frequently. For example, current core routers receive prefix updates every five minutes [10]. A different update mechanism needs to be employed when the address ranges are described as prefixes or simple intervals. However, in either case, updates may require to insert or delete address bounds that define address ranges. In the RT-VLC this can be easily achieved by updating the affected leaf node or subtree performing node splits or merges using preferably the bottom-up approach.

When address ranges are described as intervals, e.g., port ranges in packet classification, then the above simple address insertion or deletion is sufficient to add or remove an interval. On the other hand, when prefixes are used the update mechanism needs to store more information in order to keep



(a) Bottom-up, Uniform, 256b MEM width (b) Top-down worst-case, Uniform, 256b MEM (c) Top-down, average, Uniform, 256b MEM



(d) Bottom-up, Gaussian, 256b MEM width (e) Top-down worst-case, Gaussian, 256b MEM (f) Top-down, average, Gaussian, 256b MEM

Fig. 4. RT-VLC tree depth vs. Range Tree: Uniform and Gaussian distribution of 256-512k bounds considering 32, 64 and 128-bits address width and 256-bits memory bandwidth.

track of overlapping prefixes and multiple parts of a single prefix. To our advantage however is the fact that the nodes and branches of a RT-VLC can be mapped one to one to the ones of an original Range Tree which would have unlimited memory bandwidth. Hence the multiway Range Tree technique of storing and updating prefixes can be followed [6] in the RT-VLC approach as well.

IV. Evaluation

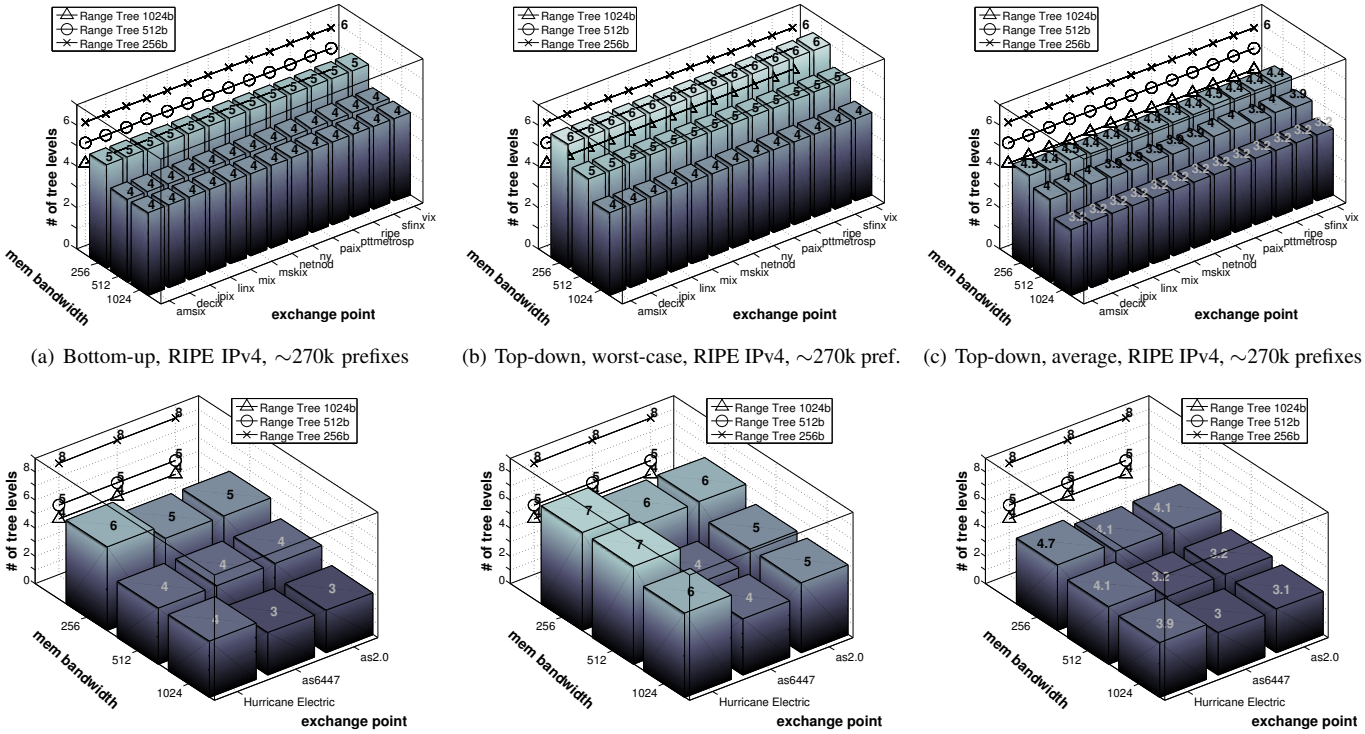
We evaluate the RT-VLC performance and scalability counting tree-levels, which correspond to memory accesses (and latency) per lookup, and the required search memory size. We use both synthetic and real IPv4 and IPv6 routing tables for constructing a RT-VLC. We compare each case with the best-case of a Range Tree[3] which in terms of lookup latency is known to scale better than trie-based solutions [1].

Figures 4 and 5 depict for the above routing tables the number of RT-VLC tree levels and compare it with the Range Tree. It is noteworthy that the bottom-up RT-VLC heuristic creates perfectly balanced trees where all the leafs are at the same level. This is not the case for the top-down heuristic and therefore we report for it both the worst and average number of tree-levels.

We generated synthetic sets of 256-512k bounds of 32, 64 and 128-bits address width that define address ranges following a uniform and also a Gaussian (variance $\sigma^2 = 2 \times$ *#of bounds*) distribution. Subsequently, we evaluate RT-VLC using the bottom-up and top-down heuristic considering 256-bits read per memory access. RT-VLC performs better for bounds with Gaussian distribution (figures 4(d), 4(e) , and 4(f)) rather than the uniform (figures 4(a), 4(b) , and 4(c)); that is because in the gaussian distribution the bounds are concentrated in a smaller range and therefore have large common prefixes. RT-VLC height and latency scales better than the Range Tree as the number of bounds increase, even for addresses of 32-bits width. Scalability improves even more as we increase the address width. Moving from 32-bit addresses to 64 and 128 bits adds one or no extra tree level for the RT-VLC, as opposed to the Range Tree which requires 3 more levels when doubling the address width for sets of 512k bounds. Finally, the RT-VLC memory requirements for 512K address ranges are for the gaussian distribution 0.9, 1.1, and 1.6 Mbytes for 32, 64 and 128 bits address width respectively, and for the uniform 3.3, 7 and 16Mbytes; that is compared to 2, 4, and 8 Mbytes that a linear search would need.

Furthermore, we used real IPv4 routing tables collected in 8/8/2008 from various locations found in [10], each one having 262,310 to 275,706 prefixes (314,756-330,445 bounds). As depicted in Figures 5(a), 5(b) , and 5(c) the top-down heuristic has similar worst-case numbers as the Range Tree, but saves more than a level on average. The bottom-up RT-VLC is one level shorter than the Range Tree which may achieve similar height only when reading 1024 bits per memory access. The memory requirements are 0.63-0.68 Mbytes for 256-bit

[3]Range Trees are constructed using B-Trees which may not maximize the number of branches per node and therefore may require more levels [6]. In our comparison we consider that Range Trees have the maximum number of branches per node allowed by the memory bandwidth.



(a) Bottom-up, RIPE IPv4, ∼270k prefixes (b) Top-down, worst-case, RIPE IPv4, ∼270k pref. (c) Top-down, average, RIPE IPv4, ∼270k prefixes

(d) Bottom-up, BGP IPv6, ∼1,6k prefixes (e) Top-down, worst-case, BGP IPv6, 1,6k pref. (f) Top-down, av., BGP IPv6, ∼1,6k pref.

Fig. 5. RT-VLC tree depth vs. Range Tree: Real routing tables of 270k IPv4, and 1.6k IPv6 prefixes, for 256, 512, and 1024 bits per memory access.

memory width, which is almost 50% that of the linear search which needs 1.2 Mbytes.

Figures 5(d), 5(e), and 5(f) illustrate the required RT-VLC and Range Tree levels for real IPv6 routing tables retrieved from Hurricane Electric, and BGP (AS6447, AS2.0) [11]. These three sets have 1631, 1597, and 1625 prefixes, respectively, which map to 2997, 2558, and 2819 unique bounds. Although, the small size of the tables reduces the benefits of RT-VLC, it still performs better than a Range Tree having in the worst case up to 40%, 20%, and 25% less levels for memory bandwidth of 256, 512, and 1024-bits per access, respectively; for the average number of levels (top-down heuristic) this improves to 50%, 40%, and 25%, respectively. The search memory requirements are about 16Kbytes, while a linear search would need more than 40Kbytes.

In general, the RT-VLC trees constructed in the above experiments have at best half the height of a Range Tree, and require for the real routing tables about half the memory size compared to linear search. In addition RT-VLC scales better than a Range Tree as the number of address ranges and address width increase.

V. Conclusions

We have introduced the RT-VLC, Range Trees with Variable-Length Comparisons, a new data structure for address lookup which performs comparisons in parts of addresses. We described four rules employed to construct a RT-VLC and reduce the required address bits to be processed per comparison. We showed the benefits in terms of latency and memory accesses per lookup, memory size and scalability. RT-VLC can store 512k address ranges of 32-128 bit addresses in up to seven tree levels which is 15-50% less levels than a Range Tree. Our experiments further showed that RT-VLC memory requirements are 0.5-2× that of the linear search algorithm. In addition, RT-VLC height scales better than a Range Tree as the routing table grows and also the address width increases. An RT-VLC that stores 512K entries needs only 2-3 more levels compared to one that stores only 256 entries, while doubling the address width may add only a single extra tree level.

References

[1] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of ip address lookup algorithms," *IEEE Network*, vol. 15, pp. 8–23, Mar/Apr 2001.

[2] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, pp. 24–32, Mar/Apr 2001.

[3] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.

[4] M. G. H. Katevenis, *The Future of Computing, essay in memory of Stamatis Vassiliadis*, ch. Interprocessor communication seen as Load-Store Instruction Generation, pp. 55–68. September 28, 2007.

[5] NRO: IPv6 Growth Increases 300 Percent in Two Years, "http://www.nro.net/documents/press_release_031108.html," Dec 2008.

[6] P. Warkhede, S. Suri, and G. Varghese, "Multiway range trees: scalable ip lookup with fast updates," *Comput. Netw.*, vol. 44, no. 3, pp. 289–303, 2004.

[7] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed ip routing lookups," in *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, (New York, NY, USA), pp. 25–36, ACM, 1997.

[8] D. R. Morrison, "Patricia—practical algorithm to retrieve information coded in alphanumeric," *J. ACM*, vol. 15, no. 4, pp. 514–534, 1968.

[9] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *IEEE INFOCOM*, pp. 1240–1247, 1998.

[10] RIPE Network Coordination Centre, "http://www.ripe.net/."

[11] "http://bgp.potaroo.net/."