Routing with a Clue

Anat Bremler-Barr Yehuda Afek Sariel Har-Peled

Computer Science Department
Tel-Aviv University
Tel-Aviv 69978, Israel
{natali,afek,sariel}@math.tau.ac.il

Abstract

We suggest a new simple forwarding technique to speed-up IP destination address lookup. The technique is a natural extension of IP, requires 5 bits in the IP header (IPv4, 7 in IPv6) and performs IP lookup nearly as fast as IP/Tag-switching but with a smaller memory requirement and a much simpler protocol. The basic idea is that each router adds a "clue" to each packet, telling its downstream router where it ended the IP lookup. Since the forwarding tables of neighboring routers are similar, the clue either directly determines the best prefix match for the downstream router, or provides the downstream router with a good point to start its IP lookup. The new scheme thus prevents repeated computations and distributes the lookup process across the routers along the packet path. Each router starts the lookup computation at the point its up-stream neighbor has finished. Furthermore, the new scheme is easily assimilated into heterogeneous IP networks, does not require routers coordination, and requires no setup time. Even a flow of one packet enjoys the benefits of the scheme without any additional overhead. The speedup we achieve is about 10 times faster than current standard techniques. In a sense this paper shows that the current routers employed in the Internet are clue-less; Namely, it is possible to speedup the IP-lookup by an order of magnitude without any major changes to the existing protocols.

1 Introduction

Motivated by the increased demand for Gigabit routers to carry the ever growing IP traffic, there have been recently several suggestions to combine fast packet switching/processing with IP routing. Specifically it is suggested to exploit the cheaper price of bandwidth compared with the price of processing. The idea is to add some information to the packet header which helps the routers along the packet path to process the packet, i.e., perform IP lookups much faster. Examples of such methods include, Tag-switching (threaded indices), IP-switching, MPLS, and source hashing [5, 16, 21, 1, 2]. In this paper we suggest *distributed IP lookup*, a new technique for IP-lookup.

The basic idea of distributed IP-lookup is that a router $R1$ sending a packet to router $R2$, adds a clue to the packet containing information on what it has learned on this packet while processing it, i.e., while processing the packet header. Router $R2$ uses the clue to start processing the packet header at the point $R1$ ended. To this end, Router $R2$ maintains a table of clues it may receive from $R1$ containing for each clue information that may help $R2$ to more efficiently process the packet. The clue helps router $R2$ to perform faster IP lookup, after which $R2$ sends the packet to router $R3$ again with a clue on what $R2$ has learned about this packet. The clue that a router includes is based only on what it has learned about the packet and is independent of the clue that came with that packet from the previous router. Notice that a router may disregard a clue, or may not include a clue on the outgoing packets - the scheme still works albeit not as efficiently as possible.

One of the most natural clues for IP routers, and the one which is considered in detail in this paper, is the best matching prefix that a router found for the packet destination address. Being a prefix of the packet destination address the clue is easily encoded by 5 bits (IPv4) indicating the part of the address which is the clue. Thus, the set of possible clues from router $R1$ to router $R2$ are the prefixes in $R1$'s forwarding table for which $R2$ is the next hop. Router $R2$ can obtain this information in one of two ways: (1) on the fly, as clues arrive, or (2) when the routing tables are being computed (by e.g., OSPF, or BGP). The information of what a router may gain from an incoming clue may either be computed for each clue when the first instant of that clue arrives, or as before, together with the forwarding tables computation. Either way, there is no need for any real time extra processing, i.e., there is no work in a new connection setup, the processing gain is achieved even if only one packet is sent in this flow (e.g., UDP). No round trip delays are incurred and no label coordination between routers, or random indices selection by the source is necessary. The extra space necessary for the clues hash table (as we will show one clue table is sufficient for all incoming links) is pessimistically about 60,000 entries (for large routers) with an average of nine bytes for each clue resulting in a total of about 540Kbyte. Another feature of our clue system is its robustness, i.e., even if neighboring routers are slightly un-coordinated the clues they send each other can not cause any confusion.

Distributed IP lookups is a natural and economical extension of IP forwarding, it performs nearly as fast as TAG-switching or threaded indices and in some cases even faster than these methods (see Section 2). Moreover, the new scheme requires fewer bits at the header, and provides a much simpler implementation. Furthermore, as our preliminary empirical tests show (see Section 6) the average number of memory references in our scheme is close to 1 (1.05 in the unfavorable case).

The Distributed IP lookup scheme divides the cost of processing a header among the routers along the packet path. Each router starts the IP lookup where its predecessor stopped. In Figure 1 we show a (*speculative*) graph of the *length* of the packet best matching prefix along a path from the source to the destination and its derivative which depicts the expected amount of work, in our method, by routers along the packet path. As can be seen from these graphs, we expect the heavily loaded routers at the heart of the Internet backbone to be the least loaded by our method.

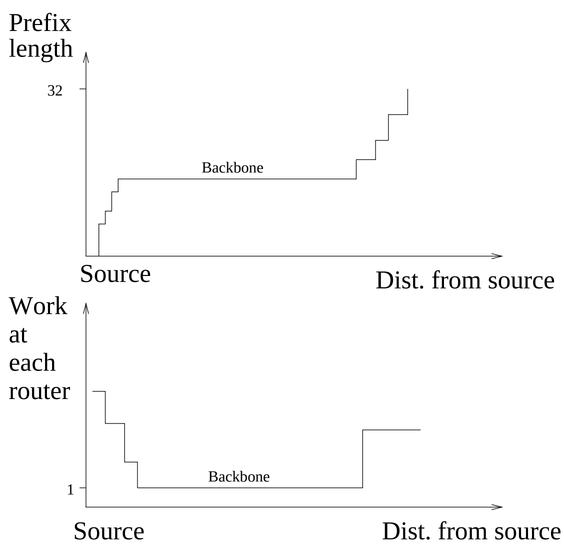

Figure 1: Best matching prefix of a packet along its way to the destination. The bottom part shows the expected amount of work by routers along the packet path.

In this paper we concentrate on different variations of distributed IP lookup and its implications. However, we believe the idea of a clue in which one router shares what it has learned from a packet with succeeding routers may have other generalizations and applications in different domains.

In the next section we compare our scheme with other related schemes. In Section 3 a detailed description of the new technique is given. Its combination with different lookup schemes is described in Section 4. In Section 5 several variations and improvements on the method and given. The various methods are experimentally analyzed in Section 6. Conclusions are given in Section 7.

2 Related work

Speeding up IP lookups is an important topic that has received considerable attention in recent years. Three major directions were taken: (1) Better implementations of the data structures and search techniques in the router, mostly software based, [23, 19, 26, 6, 11], (2) Hardware approaches to enable fast lookups with parallelism in the hardware, [12, 13], and (3) Avoiding the lookup process by adding indexing keys, such as labels, and flow identifiers in the packet headers [16, 2, 21, 1].

Data structures and algorithms: The standard IP lookup algorithm currently in use is based on radix trie (or Patricia) [22, 23]. In this implementation the prefixes are efficiently represented in a Trie (see Subsection 3.1 for definition). Each address lookup is performed by scanning the address bit by bit and matching it along a path in the trie. The worse case cost of an IP lookup is thus $O(W)$, where W is the address length (32 in IPv4, 128 in IPv6). This scheme requires $O(N)$ space, where N is the total number of prefixes in the forwarding table. The basic approaches to improve this scheme are: (1) Perform a binary search over the possible prefix lengths, requiring $O(\log W)$ steps [26]. For each test in the binary search a hash table is consulted, requiring to break the prefixes into several hash tables which all together require $O(N \log W)$ space. (2) Go over the address in different jumps, rather then bit by bit [24]. (3) Binary search over the space of N prefixes, requiring $O(\log N)$ steps [19]. This approach has been improved by relying on the SDRAM technology and performing 6-way search resulting in $O(\log_6 N)$ steps [11]. (4) Compress the prefixes data structure into the cache [6], [17]. (5) Compute locally equivalent forwarding tables that contain minimal number of prefixes [20] and hence most of the table can fit into the cache. (6) Exploit CPU caching as a hardware assist to speed up routing table lookup significantly, by treating IP addresses as virtual memory addresses [4]. (7) Minimize the average lookup time per prefix, when the forwarding table is in different memory hierarchy [3].

Hardware approach: There are several directions all based on the usage of parallelism in the hardware level: (1) Usage of pipelining to perform several lookups at the same time [8, 15]. (2) Employ low level hardware parallelism by using Content Addressable Memories (CAMs) [12, 13]. In such memories (like associative memories) the address is compared against all the prefixes in the memory in parallel. (3) Employing a cache to hold the results of recent lookups. It is possible to achieve a 90% hit rate [18, 16] but by employing a large and very expensive cache based on the CAM technology. All the hardware solutions suffer from very high costs especially when applied to large backbone routers, and they don't scale easily.

Label swapping: This direction includes IP-switching [16], TAG-switching [21] (threaded indices [2]), and MPLS [1]. Basically, the same label is attached to each packet of a flow. Routing decisions are done by one memory reference into a table of labels (similar to VC switching in ATM). Each entry in the table contains for the corresponding label, its routing decision and perhaps a new label to be swapped with the current label in the packet. The method suggested in this paper most naturally falls in this category, although it is some kind of a hybrid between the two directions, labels and efficient lookups. Furthermore, in Section 5.1 we show how the distributed routing method can be integrated with MPLS and TAG-Switching to improve their performances.

The main issue in the label swapping methods is how to associate a label to a flow, when is this association made, and may it be aggregated? Two basic approaches are: traffic (data) based label assignments, and topology (control) based label assignments. In traffic, or data, based label assignment each flow of packets receives a label, similar to VC routing in ATM. This method introduces setup overhead that delays the first packet of a flow by either a complete round trip or by just one hop in a more sophisticated implementation. Furthermore the traffic/data based method requires a relatively large number of labels, i.e., large tables in each router. In the topology/control based approach, a label is assigned to each destination or group of destinations (another more expensive possibility is to assign a label for each source destination pair, like PVC in ATM).

Either of the label approaches does not completely eliminate the need for a full IP lookup. When packets are transferred between different networks (networks that are owned by different companies) an IP lookup is required to compute new labels, in order to resolve label coordination problems. Both methods require additional coordination and communication between routers to distribute and agree on the labels. These methods require a major change in the router protocol and work only in those portions of the network that have implemented them. Since the number of labels is bounded it is impossible to assign each destination or each flow its own label. Thus, in TAG-switching for example, a label is given to a group of destinations and when the packets approach the destination they need to be separated, which requires again a full IP lookup.

In contrast to the label swapping methods, our approach uses the destination address plus a clue on each packet. It does not introduce any setup overhead, or routers coordination. The clue helps to perform the lookup much faster, sometimes as fast as in the label swapping methods. One may argue that label swapping approaches are faster since they switch the packet in $O(1)$ memory references. However, if we consider the IP lookups they require at the boundaries, and at intermediate gates, then along the entire packet path our method often incurs less processing (see Section 5.1).

Moreover, we believe that distributed IP lookup can be easily implemented in existing routers as it is a natural extension of IP routing and requires fewer changes. Furthermore, the distributed IP lookup is easy to integrate into heterogeneous networks. Even if only a few routers use the scheme, it already pays off. Mixing routes that support with routers that do not support the method does not disturb the network operation. Notice that no trust problem is created by the method since the prefixes router $R2$ learns from $R1$ are those prefixes that $R1$ has for the network of $R2$, or for destinations that are beyond $R2$ (i.e., $R2$ learns what $R1$ knows about $R2$).

3 Distributed IP lookups

In general, upon receiving an IP packet an IP router looks up in its forwarding table for the longest prefix that matches the destination address of that packet. With each prefix in its forwarding table the router keeps the next hop on the route to the destination, for all the packets for which this prefix is the longest match. In what follows we concentrate on the case in which the clue piggybacked on an IP packet sent from router $R1$ to the next router, $R2$ is the best matching prefix that $R1$ found for that packet destination address. Henceforth, the word *clue* stands for the longest prefix match that router $R1$ found for the packet destination address and send to $R2$. The word *clue* is used interchangeably for a clue, the string representing it, and the vertex in a trie that represents this string. Since the clue is a prefix of the packet destination address it can be encoded by 5 bits pointer into the destination address (IPv4). The five bits simply represent the number of leading bits of the destination address that represent the prefix. For example, for the address 125.7.19.123 and 5 bits value of 16 represent the clue 125.7.

Each router maintains a hash table of all the clues it may receive from its neighbors (see Figure 2). For each clue the table contains information that helps the router to quickly find the longest prefix of the packet destination. The information provided by a clue is essentially one of two types: either that the clue directly implies the longest prefix match of the packet destination at this router, or that a search for a longer prefix should be performed starting in a location pointed at by the clue.

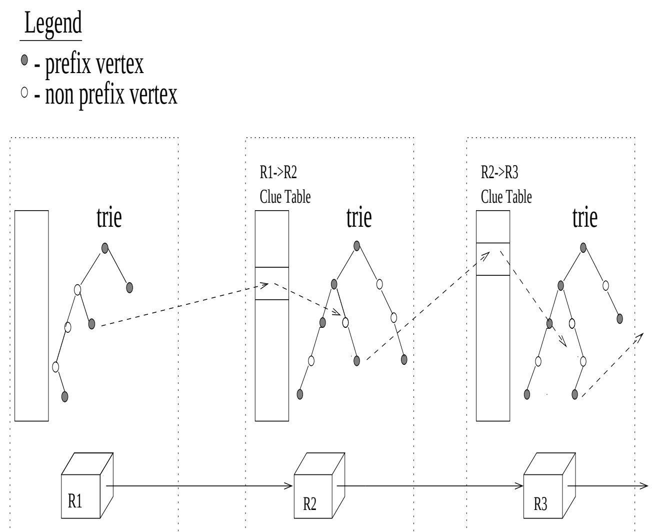

Figure 2: Schematic view of Distributed IP lookup.

The main considerations are: how useful and beneficial is the clue to the router receiving it, and what are the costs associated with the method. The cost of carrying the clue on the packet, and the cost of maintaining the clues hash table. The main effect of our method is that the longest prefix match computation is now divided (distributed) among several routers along the IP packet path.

The premise of the technique is that the forwarding tables at neighboring routers are very similar and thus in many cases the best matching prefix (BMP) found in one router is either also the BMP that is found in the next router or very close to it. There are several reasons why forwarding tables at neighboring routers are similar. One reason is simply that the computation of a forwarding table at a router is based on the forwarding tables of its neighbors and thus is strongly related to these tables. Furthermore, at certain levels of the Internet routing algorithms (BGP) aggregation of prefixes is discouraged [9] (Under BGP a router may not aggregate prefixes which it does not administer to avoid the creation of what is called "black holes". Another reason to discourage aggregation is to avoid huge changes in the routing tables following a topological change). That is, aggregation is done inside some domains, Autonomous Systems (AS), and at the borders of the AS's. Once the prefixes of destinations inside an AS are sent by the routing algorithm outside of the AS, they are not aggregated anymore with any other prefixes (by the routing algorithm). However, there are other policies carried out by BGP that may cause dissimilarities between neighboring forwarding tables. These are policies by which a BGP router tries to hide information from neighbors for policing reasons.

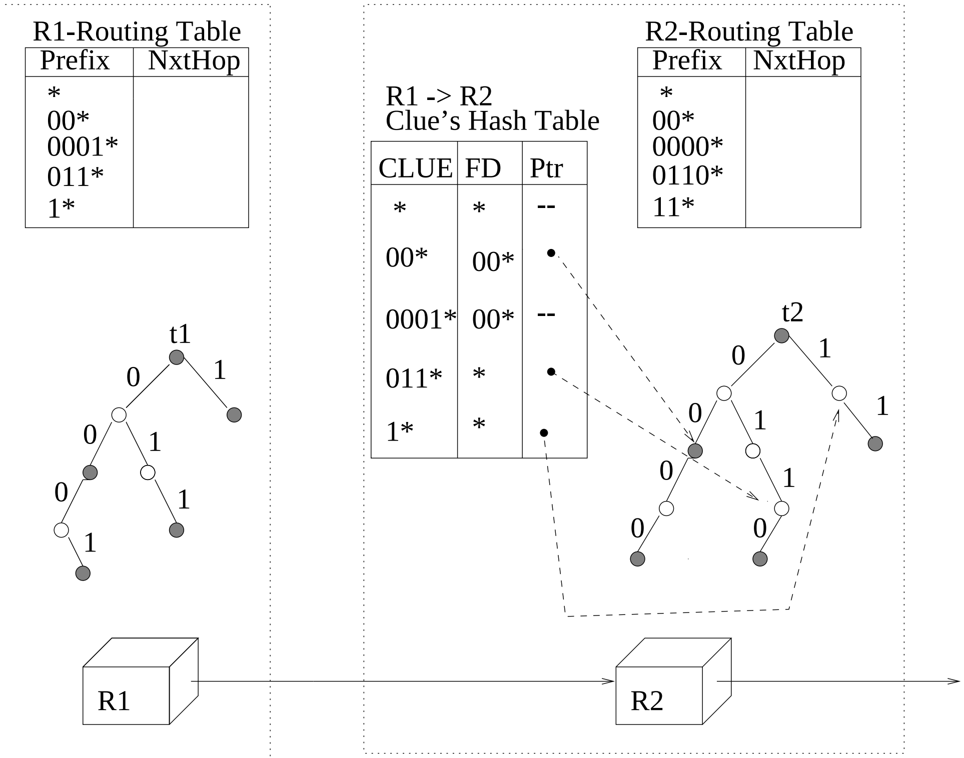

Figure 3: A more detailed view of Distributed IP lookup. For clarity we put the BMP associated with a clue's final decision (FD) and not the pointer to the forwarding table or to the corresponding trie vertex.

3.1 How to benefit from a clue

To understand how beneficial a clue is to the router receiving it, we need to analyze the relations between the tries of the neighboring routers.

A *trie* data structure is a binary tree data structure that represents all the prefixes in a router's forwarding table (see Figure 3). Each vertex in a trie represents a binary string in the natural way: The root of the tree represents the empty string. Each edge going to the left from a vertex represents 0 and an edge going to the right 1. The binary string associated with a vertex in the tree is the sequence of bits on the edges along the path from the root of the tree to that vertex. Not all the vertices in the tree represent prefixes, those that do are specially marked so. Any unmarked (non prefix) vertex in the tree that has no marked descendants is removed from the trie. Thus all the leaves of a trie are marked. In a common implementation of the trie data structure, called *Patricia*, all the internal unmarked (non-prefix) vertices that have only one child, are contracted, thus any internal vertex is either marked or has two child vertices.

As we will see, there are several cases in which router $R2$ knows by the clue alone what is the BMP of the received packet. In such cases we store in the clue's hash table either one of the following: the packet BMP, a pointer to that prefix entry in the forwarding table, or simply the next hop associated with this prefix in the routing table. Which of the above is placed in the hash table depends on the implementation and whether other decisions besides the next hop are necessary with regard to this packet. Henceforth we will denote such a value FD (final decision), which stands for either one of the above three options. Notice that placing the next hop in the clues' table requires updating the table upon changes in the routes.

Next we present two different ways, *Simple* and *Advance*, in which a router receiving a clue s may use it. *Simple* is more straightforward, requires less precomputations but does not take full advantage of the clue. *Advance*, requires a little more precomputation and takes full advantage of the clue. The expected lookup time of *Advance* is smaller than that of *Simple* (amortizing over different possible headers). For the comparison of the two methods on specific forwarding tables of large neighboring routers in the Internet see Tables 6, 7, 5, 4, 8, and 9,. Either method significantly reduces the expected processing time at the routers as can be seen in these tables.

3.1.1 Simple

In this method, upon receiving a clue s router $R2$ tries to find a longer prefix for the packet address only if in its trie vertex s has any descendants. If however vertex s has no descendants or does not exist in the trie, then router $R2$ finds in the entry of s in the clues table the best matching prefix that could possibly be found in its trie.

In this method we keep with each clue in the hash table two fields, a pointer Ptr, and an FD. If Ptr is not set to a special value, called *empty* then Ptr points to the location in the current trie that corresponds to the clue and from which the search for a longer matching prefix should continue. On the other hand, if Ptr is *empty* then no longer matching prefix may be found in this router and the FD field contains already the best matching prefix for the corresponding packet (or some other final decision as discussed before). When the Ptr is empty the best matching prefix for the packet destination is the least ancestor of s in the trie of

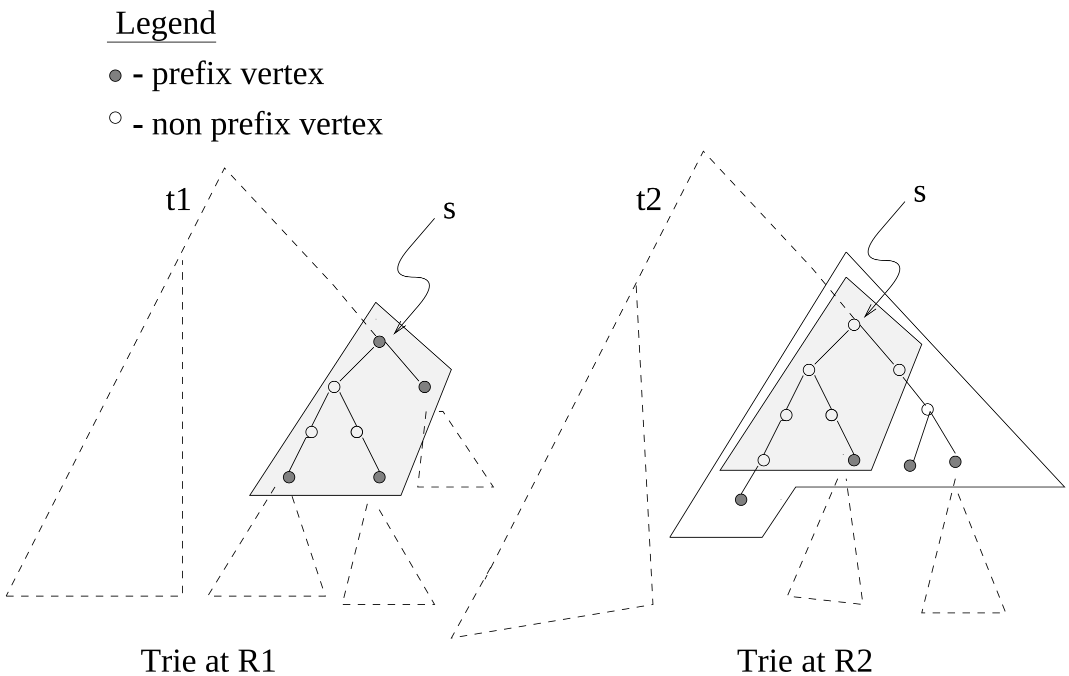

Figure 4: Conditions of Claim 1. Any path from s to a prefix in $t2$ goes through a vertex which is a prefix in $t1$.

$R2$ which is also a prefix (usually, but not necessarily, that will be vertex s by itself).

How does the search continue when Ptr is not empty depends on the implementation of the trie which is used at this router. It can be a Patricia implementation, or one of the advanced methods suggested in [26, 11, 24]. In Section 4 we discuss these possible implementations. If the search for a longer prefix fails, then the FD field contains the best matching prefix that can possibly be obtained, which is as before the least ancestor of s which is a prefix in $R2$.

3.1.2 Advance method

In this method we discover and pre-compute several more cases in which it is not necessary to continue the search for a longer prefix in $R2$ even though the vertex corresponding to clue s has descendants in the trie of $R2$. The basic claim underlying the *Advance* method is:

Claim 1 *Let s be the prefix sent as a clue from $R1$ to $R2$ on a packet whose destination address is $dest$. Let ti denote the trie data structure at router Ri. If on any path going down from s in $t2$ we encounter a prefix of $R1$ before or at the same time that we encounter the first prefix of $R2$, then no prefix of $dest$ longer than s can be found in $R2$. (see Figure 4).*

Proof: By contradiction. If the prefix $s2$ found in $R2$ is longer than (an extension of) s then by the conditions of the claim there must be a prefix $s1$ on the path from s to $s2$ in the trie of $R1$, and $R1$ should have found this longer prefix rather than s. Contradicting the fact that s is the BMP at $R1$. ∎

We farther claim that, only if the inverse of Claim 1 is satisfied, then the lookup for a longer than s prefix should continue at $R2$ (see Figure 6). I.e., only if there is at least one prefix, $s2$, extending s, in the trie of $R2$ such that no prefix on the path from s to $s2$ is also a prefix in the trie of $R1$ (including $s2$ itself, i.e., neither $s2$ is a prefix in $R1$), then a lookup for a longer prefix should continue in $R2$ (starting from s). In any other case no lookup is necessary in $R2$, the clue's hash table contains the final result for the lookup at $R2$.

Let us go over all the possible cases in the *Advance* method in detail:

Case 1- $s \notin R2$'s trie: I.e., the vertex that corresponds to s does not exist in the trie of $R2$. In this case the BMP of s, and hence of $dest$, in the trie of $R2$ is the least ancestor of s in $R2$'s trie, which is marked. Denote this ancestor as $an\text{-}s$. This ancestor may be computed off-line when the routing tables are setup. Hence, in this case we place in the hash table entry of clue s either a pointer to the entry in $R2$'s routing table that corresponds to $an\text{-}s$ or simply the next hop that is associated with $an\text{-}s$. Notice however that this case means that the prefix found in a later router is shorter than a prefix found in an earlier router. Thus, it is not expected to often show up when routing packets in the Internet.

Case 2- Claim 1 is satisfied: This case is depicted in Figure 4. Let in this case ls be the longest prefix of s which is a prefix in $R2$'s trie. If s is also a prefix in $t2$ then ls equals s. Otherwise, ls is the least ancestor of vertex s in $R2$'s trie which is also a prefix in $R2$ (i.e., ls is the BMP of s in $t2$). In either case ls or the information associated with ls, is placed in the FD field in the table. This prefix may be computed off-line when the routing tables are being constructed.

Case 3- The inverse of Claim 1 holds: In this case there is a set $\mathcal{S}$ of prefixes in $t2$ such that there is no prefix $s1$ in $t1$ longer than s and shorter or equal to any $s2 \in \mathcal{S}$. See Figure 6 for an example. This is the only case in which the search for the BMP should continue at $R2$.

```
Upon receiving packet with clue c, and destination d
                                        from R1 at R2
  Let index be the index associated with c ;
  entry := ClueHashTable[index] ;
  if (entry.CLUE == c )
  then                              {The Clue is in the Table}
     if (entry.Ptr == Empty)
     then route according to entry.FD ;
     else
         Find and use the BMP of d in the sub-trie rooted
         at entry.Ptr to route the packet ;
         If no such BMP exist use entry.FD
         to route the packet ;
  else  {The Clue is not in the Table, never saw this clue}
     route the packet according to the
     BMP of d in the trie of R2 ;
     Call procedure new_clue(c);


Procedure new_clue(c) at router R2
                     for a clue received from router R1
  index := new index, associated with c ;
  ClueHashTable[index].FD := BMP of c in the trie of R2 ;
  Let s be the vertex corresponding to c in R2 ;
  if s doesn't exist
  then ClueHashTable[index].Ptr := Empty ;
  else
     Let Pointer be pointer to s ;
     if  on every path from s to a vertex
         that corresponds to a prefix of router R2,
         a prefix of R1 is encountered
     then ClueHashTable[index].Ptr := Empty ;
     else ClueHashTable[index].Ptr := Pointer ;
```

Figure 5: Basic steps in the implementation of the distributed IP-lookup, advance method

In **case 3** a search for a longer prefix that matches *dest* is continued from the vertex corresponding to s in trie $t2$. There are two issues that have to be addressed in order to complete this search, first what efficient ways are there to perform the continued search, and second what should be done in case the search fails and no prefix of *dest* longer than s is found in $t2$.

A straightforward approach to perform the search is to continue linearly from the clue s along a path in $t2$. Notice however, that the search can stop as soon as it reaches a vertex for which Claim 1 holds. Better methods than a bit by bit scan from s, such as applying one of the techniques suggested in [26, 11, 24] are explored in Section 4.

3.2 Hash table fields

The same fields that were used in the *Simple* method, FD and Ptr are used here, in a similar way. If the search for a prefix of *dest* that is longer than s (**case 3**) fails then there are two possibilities: Either s is also a prefix in $t2$ and then s is the desired BMP, or, the least prefix that is an ancestor of s in $t2$ should be returned as the BMP of *dest* in $t2$. Field FD contains the BMP that should be returned in case the search fails (or the FD could contain the next hop which is associated with this BMP). For **cases 1** and **2** above the pointer field is left empty and the FD field contains the desired value, as described in the case analysis.

3.3 Constructing the clues' hash table

There are two basic approaches for the construction of the clues hash table (for both the *Simple* and the *Advance* methods). One is by pre-processing when the routing tables are being constructed. The second approach, which we found more attractive, is by learning the clues hash table and its fields on the fly, while the network is operating. Let us start with the latter approach. Before we proceed we point out that the hash table could be implemented using the SDRAM technology in which each cache line is of size 32 bytes. In that case it is possible to store two hash table entries in one cache line (see below). Notice that the hash table is expected to change very rarely and thus a perfect and efficient hashing function is feasible.

3.3.1 Learning and indexing the clues table

In this approach a router $R2$ (in either *Simple* or *Advance*) starts fresh with an empty clues hash table. Each new clue that arrives the router detects that the clue is new and inserts it into the hash table. There are two techniques to implement this approach one of which avoids the hashing all together and thus may farther reduce the costs associated with the packet processing.

In either of the following two learning techniques we use a field in the entry of each clue to store the value of the clue itself (which is there any how in hash tables). In this way, whenever we go to an entry in the table it is possible to check (in one assembly instruction, or in hardware) that the entry we reached indeed corresponds to the clue at hand.

Indexing technique This learning technique avoids the hashing by associating with each clue that may be sent from $R1$ to $R2$ a fixed index and consuming another 16 bits in the packet header. Each router $R1$ sequentially enumerates the clues it may send to $R2$, $\{1, 2, \ldots, M\}$. With each IP packet it forwards to $R2$, $R1$ includes a clue (encoded as before by 5 bits), and the clue's index (16 bits, assuming there will be at most 64K clues from $R1$ to $R2$). $R2$, upon receiving the clue s with index $ind(s)$ goes to entry $ind(s)$ in its sequential clues table. If string s is found in that entry, the processing continues as described above. However, if s does not match the clue that is associated with entry $ind(s)$, $R2$ updates this entry with s, the new clue (overwriting what ever was there before). At the same time $R2$ processes the values that should be given to the fields (Ptr and FD).

Notice that the indexing technique is inherently robust while still not requiring any pre-synchronization between the routers or pre-computation. Furthermore by avoiding the hash function the packet processing time is farther reduced. Another advantage is that the routers don't have to send the set of potential clues to their neighbors each time the routing tables are updated. The disadvantages of this method is that it requires 16 additional bits at the packet header, however, a smaller clues table is needed by employing standard caching techniques.

Learning the hash table: Here we trade the 16 bits required by the indexing technique with the usage of the hash function. A hashing function is applied to the clue s that $R1$ sent to $R2$. Then, s is compared against the clue associated with the hash table entry that was computed. If they match (again a check that can be done very fast in hardware or one assembly instruction) the processing continues as described before. If however the entry's clue does not match s or no such entry was prepared before, the computed entry is updated as in the indexing technique.

The advantages of this technique are as before, no pre-processing or routers coordination is necessary. Furthermore, here we use only 5 bits in each packet header. The technique is robust and adaptive to new clues. By using caching techniques the method can be made even more efficient and adaptive while consuming less space. The disadvantage compared with the indexing technique is the usage of a hash function.

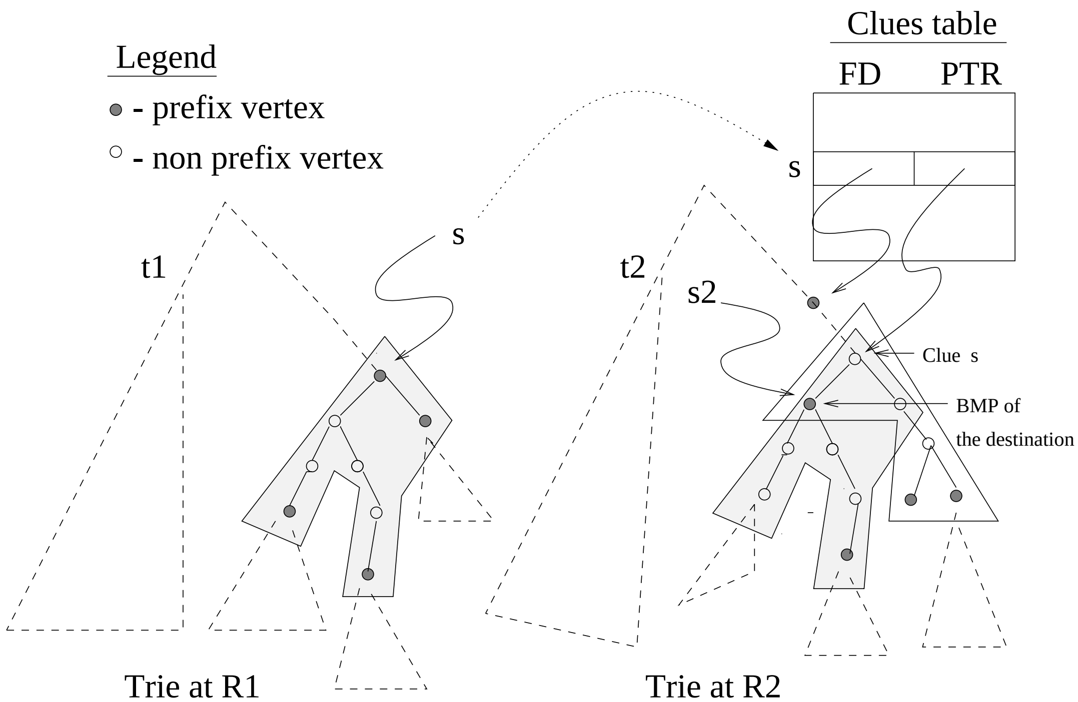

Figure 6: Conditions of the inverse of Claim 1.

3.3.2 Pre-processing construction of the clues hash table

The key idea here is that the routers will use the information they exchange in the routing algorithm (that constructs and updates the routing tables) to construct and update the clues table. Thus the actual implementation is highly dependent on the specific routing algorithm that is used, i.e., whether it is OSPF, or BGP, or both.

3.4 Combining the clues' tables of several neighbors:

A router that has several neighboring routers usually has a processor at each port that connects it with a neighboring router. In this case the hash table for each neighboring router is placed at the port. A packet arriving at the port first goes through the clues hash table. Then, the packet with the output of the clues hash table is forwarded to the switch or the routing point, depending on the specific implementation of the router/switch.

A router, that has several neighboring routers, but all its hashing tables are located in the same memory, (or several routers are connected to the same port and they all share the same clues hash tables at the port) may either treat the table with respect to each neighbor separately, or take the union of the clues it may receive from all the neighbors together. In the former we end up with several (as the number of neighboring routers) hash tables in one huge table which is not space efficient. But in this case we can take full advantage of the Advance method. In the latter, when we take the union of all the clues, the three cases above (Subsection 3.1.2) should be checked much more carefully. It might be that clue s falls into one category if it comes from one neighboring router and into a different category if it comes from another router. Specifically, for Claim 1 to apply for clue s in this case, it must hold for clue s with respect to all the neighboring routers that share the clues' table.

There are two other possibilities to preserve the small size of the clues' table with several neighboring routers:

Bit Map: Since a clue provides one of two possibilities, either it directly implies the BMP, or to continue the search, we may add to each clue a bit map of size d, where d is the number of neighboring routers. Notice that if the clue implies the BMP for several routers, then it implies the same BMP to all of them. For each clue arriving from neighboring router j we first examine the j's bit and then decide how to proceed.

Sub-tables: Here we suggest to maintain several tables, one with the clues common to all the routers and for which the behavior with regard to all neighbors is the same, and then a specific table for each neighbor. An arriving clue has to be looked in both the common table and in the specific table of the router from which the clue came. Depending on where the clue was found and with what values, the processing of the lookup continues.

Another way by which we could make the clue scheme more efficient is by insisting that a clue is never removed from a clues' table (this requires a special marking for clues that are not valid). This makes the hash function stable and more efficient and minimizes the overhead due to topological changes. A clue in the clues' table which is not in use does not disturb, except for the space it consumes in the memory (which could be ignored if caching is used). Notice however that inactivating or activating a clue requires, in the *Advance* method, updates of other fields in the clues' table.

3.5 Clues hash table space requirements

A pessimistic bound on the clues hash table size assumes that the number of entries in the hash table is about the same as the number of entries in the routing table of a large router (60,000), and that each entry requires the maximum space of three 4 bytes fields, FD, Ptr, and the clue value. However, in the *Advance* method only clues for which Claim 1 does not hold require the Ptr field. The empirical tests show that the fraction of these entries is less than 10%. Altogether, we get about 500K-600K byte. This size does not even double the space requirements of the fast memories in the current routers. Furthermore, parts of the clues hash table can be cached and placed into the cache only if touched recently. As mentioned above the table could be placed in a SDRAM cache in which each line is 32 bytes long and in one memory reference the whole record of two clues is fetched. We omit the discussion of these caches from this paper.

4 Integration with different data structures

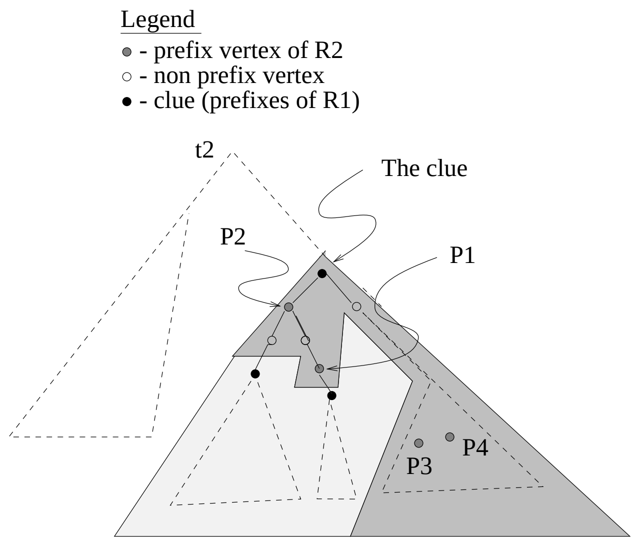

Figure 7: The prefixes over which the search continues when Claim 1 does not hold (in the darken area).

There are several different implementations of the lookup in current routers and in the literature. The distributed IP lookup method may work with either of them. However, our method may take further advantage of several of these methods when the lookup continues from the location pointed by the clue. As can be seen in Figure 7 the space over which the lookup for a BMP should continue is restricted due to Claim 1. The potential prefixes in $t2$, the trie of $R2$, that may still be BMP's of the current packet, given the clue s, are those by which Claim 1 is violated. For example, in Figure 7 only prefixes $P1$, $P2$, $P3$, and $P4$ from those shown in the figure are potential BMPs of the current message. More specifically:

Definition 1 **Condition C1:** *(See Figure 7) Any prefix p in $t2$, which:*

1. *Is a descendant of s, and*
2. *Except for s there is no other prefix in $t1$ on the path from s to p,*

might in R_2 be a BMP of the current packet destination address.

Below we describe how different lookup techniques may be adapted to the special case of looking up a BMP given its clue s:

Adapting Patricia: Set s to be the initial BMP. The lookup proceeds by simply walking on the Patricia trie from the clue s until the walk cannot continue as dictated by the sequence of bits in the destination address (because the corresponding branch is missing). Notice, that automatically the walk never reaches a prefix which is also a prefix in $t1$ (otherwise, that prefix would have been found by $R1$). The last prefix that was encountered by the walk is the desired BMP. Notice that we can further improve the search by applying Claim 1 to each vertex in the Patricia trie. We associate with each vertex a Boolean indicating whether the search should continue from this vertex or not (a knowledge that can be acquired by the application of Claim 1 to these vertices). Whenever the search reaches a vertex from which it should not continue, the desired BMP is the last prefix that was encountered. If a router has several neighboring routers, then we have to add one such Boolean bit at each vertex for each neighboring router.

Adapting binary search: The set of potential prefixes given a clue s that arrives from router $R1$, is denoted $\mathcal{P}$ $(s, R1)$. The search for the BMP of the packet destination is performed using a standard binary search. In general, the set $\mathcal{P}$ is expected to be small, e.g., just a few prefixes. In such a case the entire set, may be placed in the same cache line with the clue's entry in the table (if we use SDRAM then each entry could contain few prefixes in addition to the other fields of an entry). When the entry is fetched, the corresponding potential clues are brought into the cache line, and the appropriate prefix is found without any further external memory accesses. If however the set $\mathcal{P}$ is larger, then the *6-way* method [11] may be employed. The *6-way* method is the same as the binary lookup but on the basis of 6-way branching rather than binary branching.

Adapting the $\log W$ method: Given the set $\mathcal{P}$ it is possible to determine what is the minimum length and maximum length of a possible BMP in $\mathcal{P}$. Given these lengths it is possible to adapt the method of [26] to perform a binary search over the range of possible BMP length. In each step of the search we check for a given length i whether a string longer than the length i prefix of the destination address is a possible BMP. If yes, we step forward according to the current step size in the binary search, and if not then either the length i prefix is the desired BMP or we step backward according to the current step size in the binary search.

5 Variations and farther improvements

5.1 Integrating clue routing with MPLS and Tag-Switching

In one of MPLS variants, that concerning topology (control) based label assignments, a label is bound to a prefix and all the packets whose best matching prefix (BMP) is the same carry the same label and are switched using this label [7]. The same technique is used in Tag-Switching, and hence all that is mention below for MPLS is valid also to Tag-Switching.

When such a stream of packets arrives at a router whose forwarding table contains one or more prefixes *that extend* the prefix that was bound to this label, this router has to perform an IP-lookup on the packet destination address to decide on which outgoing port and with which new label to forward the packets in the stream (see Figure 8). This is the point where the distributed IP-lookup method can be combined with MPLS.

Notice that each label in MPLS (control based) is associated with a clue in distributed IP-lookup (since the label is associated with a specific prefix). I.e., each label implies the clue that would go with the corresponding packet. Hence, the label can be used as an efficient indexing into the clues table, thus eliminating the hash function in this combination. Therefore, the downstream MPLS router that performs IP-lookup (e.g., router $R4$ in Figure 8) on this packet destination address would use the clue associated with the label to considerably expedite the lookup process (since the router would use the clue to efficiently perform the lookup as described in Section 3).

5.2 BGP over OSPF and other considerations

In the Internet it is often the case that a packet is sent from one (BGP) router to another across either an AS (Autonomous System) that internally uses OSPF routers or over a different network such as an ATM network. We claim that the distributed IP lookup scheme is still useful in that scenario.

In such cases the router goes twice through its forwarding table [9]. In the first time it finds the next hope is the BGP router on the other side of the AS but no interface port is associated with this BMP. It then takes the IP address of this router and goes with it for a second time through the forwarding table to find out what is the next hop in the AS on the route to the BGP router on the other side. In such a case the clue it places on the packet is still the first BMP it finds, since any successive router starts by looking for the BMP of the packet destination address. In some cases it might be beneficial to place both BMPs on the packet.

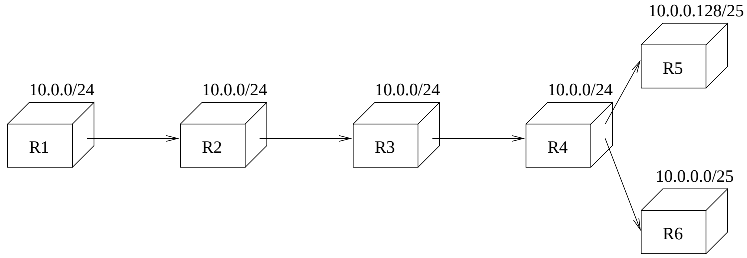

Figure 8: Aggregation point (Router R4) in MPLS. Router R4 has to perform IP-lookup for packets arriving from $R3$ with the label which is associated with the prefix 10.0.0/24 (/24 means that this prefix is 24 bits long) [5].

5.3 Integration with existing routers

The scheme suggested herein is easily integrated into a heterogeneous IP network that consists of different IP routers. Moreover, the information one router needs from its neighboring router is expected to be harmless. There are several reasons for that:

1. No coordination between neighboring routers is necessary. Using the *Simple* method with a hash table for the clues and learning the table on the fly requires no coordination between neighboring routers at all, not in the pre-computation stage and not in real-time when the flow of packets goes through. The most coordination that may be required is in the *Advance* method with indexing into a sequential table. This combination requires that any router, $R1$, would be able to deduce all the prefixes of its neighbors for which $R1$ might be the next hop. Given the information exchanged between neighboring routers during the routing algorithm, it seems possible to add the information router (e.g., $R1$) needs to this exchange. Notice, that this is necessary only in the pre-computation stage and only if we don't use the learning method.
2. A router that does not employ the distributed IP lookup method does not prevent other routers from using it. If a router participates in the distributed IP lookup then, as long as there is another upstream router that participates in the scheme, they may both benefit each other (assuming that intermediate routers relay the clue). Of course the closer they are the more they expect to help each other. Even if the packet has traveled several hops since a clue was last added to it, the clue it carries is still a prefix of the packet destination and could save a distant router some of the processing. The amount of savings depends on many factors, it could still be that the clue is the best matching prefix for the packet, or that the clue is much shorter than the BMP of the packet at the current router.
3. The distributed IP lookup scheme works well with any implementation of the IP network layer. That is, it is easily integrated into IP routers of different vendors that co-exist in the same network today. It is quite possible that the 5 bits find their place in the current IP header, e.g., in the options field.
4. **Sensitivity of routing information:** In many cases it could be that managers would hesitate to integrate the scheme since one router supposedly learns the prefixes of the other. Here we argue that this worry is not justified. Moreover, we show how it can be subdued.
    (a) In the distributed IP lookup scheme, a router need to learn only those prefixes of its neighbor for which it is the next hop. These are prefixes of information that goes in its direction anyway. Any other prefix of its neighbor it will not learn and does not need.
    (b) A router may either refrain from sending some clues (prefixes), or may truncate some clues. The scheme is still beneficial for the other clues. Truncated clues are also beneficial, perhaps not as much as the original.

5.4 Load balancing and farther improvements

Here we argue that the distributed IP lookup method can be used as a tool to balance the work load between routers. So far the scheme was described as a mechanism that is added to the existing IP routing mechanisms and makes them work faster. Here we ask what if we now use the IP routing tables together with the clue mechanism to shape the work load distribution along different paths in the network.

For example, let us guarantee that all the clues that may be sent from large back bone router $R1$ to its neighboring large router $R2$ are prefixes at $R2$ which may not be extended any farther. Then, router $R2$ performs IP lookup for each packet arriving from $R1$ in one memory reference, just as in TAG-switching (but does not need to swap the label/clue). In a more aggressive implementation of this idea one could shape the work load across the network. In such an approach the work load of heavy traffic backbone routers is minimized while the peripheral and edge routers are required to gradually lookup for longer and longer prefixes. Notice that aggregation was carefully implemented in the network in a way that does not create routing loops (over aggregation could potentially create such loops). However, our suggestion here amounts at reducing the aggregation rather than increasing it and hence may not create routing loops.

6 Empirical tests

In Tables 6, 7, 5, 4, 8, and 9, we compare the number of memory accesses (i.e., number of steps) required by different IP lookup methods. To perform the experiments we took snapshots of the forwarding tables of the following routers at about the same time, (either from [14] or using "sh ip route"): { MAE-East, MAE-West, Paix, and the two couples of routers (AT&T-1, AT&T-2) and (ISP-B-1, ISP-B-2). Each such couple is large neighboring routers in a large ISP (AT&T and ISP-B respectively) }. The first three tables are route-server tables while the last four are actual forwarding tables. We then compared many different pairs of these routers (we used several other routers, but the results are similar to those reported here). For each pair we simulated 10,000 packets with different destinations going from one router to the other. For each of these we counted the number of memory accesses (to a table or the trie) that are made at the receiving router.

The destination addresses for the experiments were selected as follows: Let $R1$ be the sending router and $R2$ the receiving router. A random destination is chosen, and its BMP in $R1$ is computed. Then we verified that this BMP is a vertex in the trie of $R2$, and if so the processing of that packet at $R2$ was carried out. If the BMP is not a vertex then this destination was not considered in our experiment. This was done in order to predict for which selected destinations router $R2$ is a possible next hop. (Not all the routers we checked were immediate neighbors, and the knowledge of next hop was problematic, though the two router pairs of the ISPs are immediate neighbors). Certainly, eliminating these destinations from our experiments does not make our results look better. Since, if the BMP (which is the clue sent from $R1$ to $R2$) is not a vertex in the trie of $R2$ the clues' table immediately

MAE-East	MAE-West	Paix	AT&T-1	AT&T-2	ISP-B-1	ISP-B-2
42,250	24,123	5,974	23,414	60,475	56,034	55,959

Table 1: Total number of prefixes in each table.

		Equal Clues
MAE-EAST	MAE-West	23,382
MAE-EAST	Paix	5,899
MAE-WEST	Paix	5,814
AT&T-1	AT&T-2	23,381
ISP-B-1	ISP-B-2	55,540

Table 3: The total number of prefixes of one router that also appear in the other (i.e., the intersection size).

provides the desired lookup, at the minimum cost of one memory access (to the clues' table).

Sender	Receiver	Problematic Clues
MAE-EAST	MAE-West	288
MAE-EAST	Paix	35
Paix	MAE-East	411
AT&T-1	AT&T-2	575
AT&T-2	AT&T-1	52
ISP-B-1	ISP-B-2	66
ISP-B-2	ISP-B-1	38

Table 2: The total number of different clues that the sender may send and for which Claim 1 does not hold at the receiver. We call these kind of clues "problematic clues"

For each pair of routers we counted the average number of memory accesses performed by the 10,000 packets sent from one to the other under the following lookup schemes (see the tables). Each of the experiments was repeated several times and the repeated results were extremely close to each other. Five basic methods for looking up a best matching prefix were considered: (1) *Regular* which is a bit by bit scan of the destination to find the matching point in the trie, (2) *Patricia* [23, 22] which is an efficient implementation of the trie (see Section 4), (3) *Binary* [19] see the description in Section 4, (4) *6-way* [11] which is the same as the binary lookup but on the basis of 6-way branching rather than binary branching, (5) *Log W* [26] which is described in Section 2 and 4. We compared 15 different ways of performing the lookup: The basic five above without the clue, this set is called common in the tables. Our *simple* method combined with each of the above five, i.e., when a lookup has to be performed from the clue, the corresponding method was applied within the sub-tree rooted at the clue (see Figure 7). And, our *Advance* method combined with each of the above five (see Section 4).

The most impressive result from all our experiments (including many other that are not documented here) is that using the *Advance* method combined with any lookup scheme results in near optimal number of memory accesses, 1. The minimum number of memory accesses is one since each IP lookup requires at least looking up the clue in the clues' table. This minimum also applies to MPLS/TAG-switching which also need at least to lookup the label in the labels' table. The combination of the *Advance* method with Patricia or the 6-way method are slightly better. The main reasons for the good results are first that the forwarding tables of neighboring routers are very similar, and furthermore Claim 1 applies to a vast majority of the clues sent from one to the other (95% to 99.5%). Notice that the *Advance* method is about 22 times better than the simple trie scheme, and 3.5 times better than the Log W technique of [26]. Moreover, the presented scheme is expected to give similar performances in IPv6 while the Log W technique does not scale as good [11] (assuming IPv6 uses aggregation in a way similar to IPv4).

While in routers in which prefixes are not aggregated (Claim 1 holds) both our method and MPLS/TAG-switching require one table lookup, at points of aggregation our method works more efficiently since we use the clue, while MPLS/TAG-switching perform a complete standard IP-lookup to determine the new label. As mentioned before our method may be combined with MPLS to achieve the other advantages of MPLS together with the efficiency of our method.

If one uses the *Simple* method rather than the *Advance* method, she/he still gets a considerable performance gain (about 10 times better than the standard methods, and about 50% improvement over the Log W method (when compared against Simple with Patricia, for example)). Moreover, not only this scheme is more space efficient and simple to implement, it is also expected to nicely scale in IPv6.

Notice that the combination of the *Advance* method with Patricia (or trie) is better than its combination with Log W or the binary method. We believe the reason for that is that the former searches more locally while the later jumps all over the search space. This, together with the fact that the clue brings us close to the point where the search stops gives the combination with Patricia an advantage.

7 Conclusions

We have presented the distributed IP lookup scheme which considerably speeds up IP lookup with little overhead. The scheme is a natural extension of IP routing, and works at least as efficient as MPLS/TAG-switching.

Distributed IP lookup can support and be beneficial for other current and future IP services such as: IP-multicasting, and IP packet filtering [10, 25]. For example, when a packet header is classified by several filters (in QoS, or firewall applications), the clue being added to the packet is the filter by which the packet is classified at a router. The receiving router start its classification process at the restricted domain of the clue-filter. Moreover, similarly to Claim 1, any filter that both routers have and that intersect the clue-filter can be discarded by $R2$ without any processing.

Acknowledgments:

The authors are grateful to Craig Labovitz from Merit Network Inc., Michael Merritt and Fred True from AT&T for their critical help in providing us with very important snapshots of routing tables. We would like also to thank Oren Singer from CISCO Israel for his help in reading and understanding routing tables.

References

[1] Callon, R., Doolan, P., N.Feldman, Fredette, A., and G.Swallow. A framwork for multiprotocol label switching. Tech. rep., IETF, November 1997. draft-ietf-mpls-framework-02.txt.

[2] Chandranmenon, G., and Varghese, G. Trading packet headers for packet processing. *IEEE Transactions on Networking* (April 1996).

[3] Cheung, G., and McCanne, S. Optimal routing table design for ip adress lookups under memory constraints. In *Proc. INFOCOM 99* (March 1999).

[4] Chiueh, T. C., and Pradhan, P. High performance ip routing table lookup using cpu caching. In *Proc. INFOCOM 99* (March 1999).

[5] Davie, B., Doolan, P., and Rekhter, Y. *Switching in IP Networks*. Morgan Kaufmann Publishers Inc., 1998.

[6] Degermark, M., Brodnik, A., Carlsson, S., and Pink, S. Small forwarding table for fast routing lookups. In *Proc. ACM SIGCOMM 97* (Octeber 1997).

[7] E.C. Rosen, A. V., and Callon, R. Multiprotocol label switching architecture. Tech. rep., IETF, 1999. draft-ietf-mpls-arch-05.txt.

[8] GUPTA, P., S.LIN, AND MCKEOWN, N. Routing lookups in hardware at memory access speeds. In *Proc. INFOCOM* (April 1998).

[9] HALABI, B. *Internet Routing Architectures.* New Riders Publishing, Cisco Press, 1997.

[10] LAKSHMAN, T., AND STILIADIS, D. High speed policy-based packet forwarding using efficient multidimensional range matching. In *Proc. ACM SIGCOMM 98* (Sept 1998).

[11] LAMPSON, B., SRINIVASAN, V., AND VARGHESE, G. Ip lookups using multiway and multicolumm search. In *Proc. INFOCOM 98* (March 1998).

[12] MCAULEY, A., AND FRANCIS, P. Fast routing table lookup using cams. In *Proc. INFOCOM* (March-April 1993), pp. 1382–1391.

[13] MCAULEY, A. J., TSUCHIYA, P. F., AND WILSON, D. V. Fast multilevel hierachical routing table using content-addressable memory. In *Proc. SIGCOMM 95* (January 1995). U.S. Patent serial number 034444. Assignee Bell Communications research Inc Livingston NJ.

[14] Ipma statistics. http://nic.merit.edu/ipma.

[15] N. F. HUANG, S. M. ZHAO, J. Y. P., AND SU, C. A. A fast ip routing lookup scheme for gigabit switching routers. In *Proc. INFOCOM 99* (March 1999).

[16] NEWAN, P., MINSHALL, G., AND HUSTON, L. Ip switching and gigabit routers. *IEEE Communications Magazine* (Junary 1997).

[17] NILSSON, S., AND KARLSSON, G. Fast address look-up for internet routers. In *Proc. IEEE Broadband Communications 98* (April 1998).

[18] PARTRIDGE, C. Locality and route caches, February 1996. NFS Workshop on Internet Statistics Measurement and Analysis.

[19] PERLMAN, R. *Interconnections, Bridges and Routers.* Addison-Wesley, 1992, 1992.

[20] R. P. DRAVES, C. KING, S. V., AND ZILL, B. D. Constructing optimal ip routing tables. In *Proc. INFOCOM 99* (March 1999).

[21] REKHTER, Y., DAVIE, B., KATZ, D., ROSEN, E., SWALLOW, G., AND FARINACCI, D. Tag switching architecture overview. Tech. rep., IETF, 1996. ftp://ds.internic.net/internet-drafts/draft-rfced-info-rekhter-00.txt.

[22] S. LEFFLER, M. MCKUSICK, M. K., AND QUARTERMAN, J. *The Design and implementation of the 4.3BSD UNIX.* Addison-Wesley, 1988.

[23] SKLOWER, K. A tree-based routing table for berkeley unix. Tech. rep., 1992.

[24] SRINIVASAN, V., AND VARGHESE, G. Faster ip lookups using controlled prefix expansion. In *Proc. ACM Sigmetrics 98* (June 1998).

[25] SRINIVASAN, V., VARGHESE, G., SURI, S., AND WALDVOGEL, M. Fast and scalable layer four switching. In *Proc. ACM SIGCOMM 98* (Sept 1998).

[26] WALDVOGEL, M., VARGHESE, G., TURENER, J., AND PLATTNER, B. Scalable high speed ip routing lookups. In *Proc. ACM SIGCOMM 97* (Octeber 1997).

AT&T-1	Method	Trie	Patricia	Binary	6-Way	LogW
AT&T-2	Common	23.5899	20.7928	17	7	3.448
	Simple	2.0801	2.0565	2.1189	2.0456	3.0447
	Advance	1.0552	1.0442	1.0490	1.0190	1.0519

Table 4: Average number of memory accesses for packets processed by AT&T-2 after being received from AT&T-1.

AT&T-2	Method	Trie	Patricia	Binary	6-Way	LogW
AT&T-1	Common	23.2410	18.8685	16	6	3.6339
	Simple	2.0583	2.0396	2.0947	2.0367	3.0566
	Advance	1.0011	1.0011	1.0011	1.0004	1.0029

Table 5: Average number of memory accesses for packets processed by AT&T-1 after being received from AT&T-2.

ISP-B-1	Method	Trie	Patricia	Binary	6-Way	LogW
ISP-B-2	Common	23.7377	20.0929	17	7	3.3135
	Simple	2.0683	2.0467	2.1014	2.0393	3.0447
	Advance	1.0044	1.0031	1.0040	1.0015	1.0045

Table 6: Average number of memory accesses for packets processed by ISP-B-2 after being received from ISP-B-1.

ISP-B-2	Method	Trie	Patricia	Binary	6-Way	LogW
ISP-B-1	Common	22.7328	20.1494	17	7	3.3272
	Simple	2.0746	2.0574	2.1223	2.0474	3.0543
	Advance	1.0025	1.0022	1.0024	1.0009	1.0026

Table 7: Average number of memory accesses for packets processed by ISP-B-1 after being received from ISP-B-2.

Paix sends clue to	Method	Trie	Patricia	Binary	6-Way	LogW
MAE-East	Common	22.5483	19.7820	17	7	3.4705
	Simple	2.0647	2.0436	2.0897	2.0347	3.0387
	Advance	1.0617	1.0497	1.0590	1.0228	1.0599
MAE-West	Common	22.553	19.0658	16	7	3.5063
	Simple	2.0476	2.0336	2.0732	2.0283	2.0356
	Advance	1.0391	1.0320	1.0394	1.0152	1.0445

Table 8: Average number of memory accesses for packets processed by Router MAE-East and MAE-West after being received from Paix.

MAE-East sends clue to	Method	Trie	Patricia	Binary	6-Way	LogW
MAE-West	Common	22.7288	19.0678	16	7	3.5295
	Simple	2.0500	2.0367	2.0367	2.0150	3.0471
	Advance	1.0070	1.0057	1.0069	1.0026	1.0073
Paix	Common	22.4478	17.2646	14	6	3.5277
	Simple	2.0544	2.0404	2.0924	2.0358	3.0508
	Advance	1.0097	1.0090	1.0114	1.0044	1.0097

Table 9: Average number of memory accesses for packets processed by Router MAE-West and Paix after being received from MAE-East.