Efficient entry-reduction algorithm for TCAM-based IP forwarding engine

P.-C. Wang, C.-T. Chan, R.-C. Chen and H.-Y. Chang

Abstract: Ternary content-addressable memory has been widely used to perform fast routing lookups. It is able to accomplish the best matching prefix searching in $O(1)$ time without considering the number of prefixes and their lengths. As compared to the software-based solutions, the ternary content-addressable memory can offer sustained throughput and simple system architecture. However, it also comes with several shortcomings, such as the limited number of entries, enormous cost and power consumption. Accordingly, an efficient algorithm is proposed to reduce the required size of ternary content-addressable memory. The proposed scheme can eliminate 98% of ternary content-addressable memory entries by adding comparatively little DRAM and, thus, is attractive for IPv6 routing lookup.

1 Introduction

To forward packets toward their destinations, a router must perform forwarding decisions based on the routing prefixes gathered by the routing protocols. Given an incoming packet, the routing lookup operation finds the longest prefix in the forwarding table that matches the destination address. Since the development of CIDR in 1993 [1], the variable-length routing prefixes incur the best matching prefix (BMP) problem. It may be time consuming to determine the BMP for a router with a large number of table entries.

There have been remarkable proposals in the organisation of forwarding tables during the last few years. The proposals include solutions for both hardware and software. However, hardware-based schemes [2–4] generally cannot support IPv6 because of their inflexible data structures, while software-based schemes [5–7] are unable to keep up with the swift updating rate owing to their burdensome precomputation costs.

Features of TCAM open up new possibilities, particularly for the BMP problems. TCAM devices can prioritise search results to resolve the multiple matches, corresponding to different prefix lengths, which is in accordance with BMP requirements. For each destination address, the TCAM performs searching within all the prefixes in parallel. Since several prefixes may match the destination address, a priority encoder then selects the first matching entry as the result. Note that the resulting address automatically coincides with BMP entry address because of the way in which prefixes are stored. With the compelling technical advantages, TCAM-based networking devices become a preferred solution for fast, sophisticated IP packet forwarding.

Even though the applications of TCAM technology are gradually growing, it still comes with some shortcomings. For example, TCAM operates with a lower clock rate, but accompanies much higher power consumption and thermal dissipation. And also, TCAM is very expensive due to its larger chip area.

Table management is another issue of TCAM. As described above, the prefixes in the TCAM are listed in a sorted order. However, the routing table is dynamic; prefixes can be inserted or deleted due to the changes in network status. These changes can occur at a rate as high as 1000 prefixes per second [8], and hence it is desirable to maintain quick TCAM updates and keep short updating time intervals. In [8], Shah and Gupta proposed two update algorithms. By keeping all the unused entries in the center of the TCAM, each prefix insertion/deletion can incur prefixes swapping between different lengths. The worst-case update time is $W/2$.

In general, reducing the number of TCAM entries can improve the adaptability in terms of power consumption, price and board area. In this study, we introduce a novel TCAM entry-reduction algorithm. The new scheme can reduce the required TCAM entries by 98%. Furthermore, the reduced entries also eliminate the number of different lengths to improve the update speed.

2 Related work

Extensive studies have been carried out in constructing the routing tables during the past few years. The proposals include both hardware and software solutions. In [2], Degermark *et al.* use a trie-like data structure. The main idea of their work is to quantify the prefix lengths to levels of 16, 24 and 32 bits and expand each prefix in the table to the next higher level. It is able to compact a large routing table with 40 000 entries into a table with size 150–160 kbytes. The minimum and maximum numbers of

IEE Proceedings online no. 20041153

doi:10.1049/ip-com:20041153

Paper first received 7th April and in revised form 11th August 2004

P.-C. Wang is with the Institute of Computer Science and Information Technology, National Taichung Institute of Technology, Taichung, Taiwan 404, Republic of China

C.-T. Chan is with the Telecommunication Laboratories, Chunghwa Telecom Co. Ltd., Taipei, Taiwan, Republic of China

R.-C. Chen is with the Department of Logistics Engineering and Management, National Taichung Institute of Technology, Republic of China

H.-Y. Chang is with the Department of Information Management, I-Shou University, Kaohsiung, Taiwan, Republic of China

E-mail: abu@ntit.edu.tw

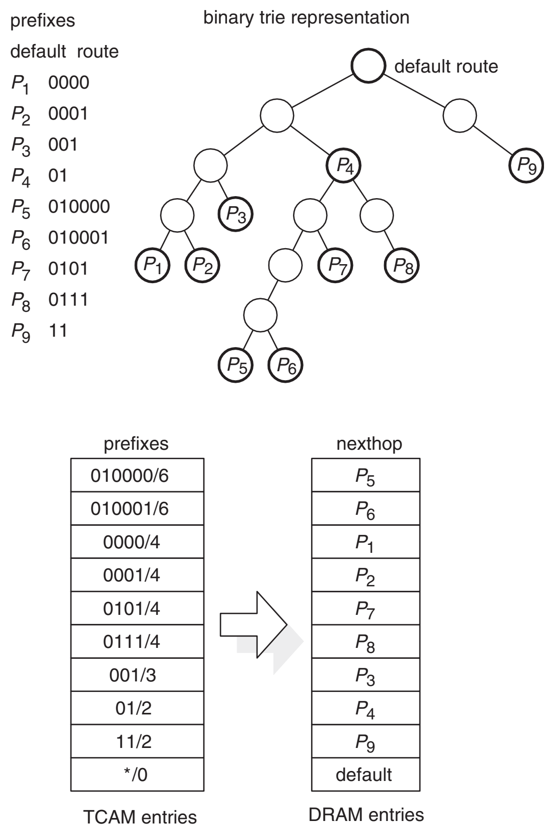
memory accesses for a lookup are two and nine, respectively, in hardware implementation. Gupta *et al.* present fast routing-lookup schemes based on a huge DRAM [3]. The scheme accomplishes a routing lookup with the maximum of two memory accesses in the forwarding table of 33 Mbytes. By adding an intermediate-length table, the forwarding table can be reduced to 9 Mbytes; however, the maximum number of memory accesses for a lookup is increased to three. When implemented in a hardware pipeline, it can achieve one route lookup for every memory access. This furnishes approximately 20 million packets per second (MPPS). In addition, Wang *et al.* derived the scheme further by fitting the forwarding table into SRAM [4].

Regarding software solutions, algorithms based on tree, hash or binary search have been proposed. Srinivasan and Varghese [9] present a data structure based on binary tree with multiple branches. By using a standard trie representation with arrays of children pointers, insertions and deletions of prefixes are supported. However, to minimise the size of the tree, dynamic programming is needed. In [5], Nilsson and Karlsson solve the BMP problem by LC tries and linear search. Waldvogel *et al.* propose a lookup scheme based on a binary search mechanism [6]. This scheme scales very well as the size of address and routing tables increases. It requires a worst-case time of $\log_2(\textit{address bits})$ hash lookups. Thus, five hash lookups are needed for IPv4, and seven for IPv6 (128-bit). The software-based schemes have been further improved by using multiway and multicolumn search techniques [7, 10]. Although these approaches feature certain advantages, they either use complicated data structures [7, 9, 10] or are not scalable for IPv6 [2–5].

3 TCAM entry-reduction algorithm

Our idea is to merge multiple routing prefixes into one representative prefix. The prefixes generated by the proposed algorithm are inserted into the TCAM and the original prefixes are stored in the extra memory, such as DRAM. The routing lookup will consist of one TCAM and intermediate table access. Assuming that the intermediate table is allocated in the DRAM, thus one additional DRAM access is required as compared to the original lookup procedure. However, we can reduce the design cost and complexity by utilising small TCAM and cheap DRAM. To begin with the proposed algorithm, we have to construct the prefix trie from the routing prefixes. The prefix trie is a binary trie constructed according to the bit-streams of the routing prefixes. As shown in Fig. 1, there are ten prefixes in the routing table, including the default route. Thus the prefixes will occupy ten TCAM and ten DRAM entries to represent the complete routing information.

The proposed subtrie construction algorithm is a recursive function and selects the roots of the subtries from the nodes of the prefix trie. It will traverse the prefix trie in the depth-first search (DFS) order. To minimise the number of subtries, we adopt the bottom-up manner to construct the subtries. While the procedure reaches a node, it will check the distance to the deepest successive leaves. If the distance is equal to the height of subtrie, say H, a subtrie rooted at the current node is generated. Otherwise, the procedure is recursively called for both children and followed by a depth check for uncovered leaves. It ensures that the depth of each generated subtrie is H. An array with 2^H entries is used to represent an H-bit subtrie for fast accessing. The addresses and depths of the subtries are recorded in the intermediate table for subtrie accessing. We



prefixes	
default route	
P_1	0000
P_2	0001
P_3	001
P_4	01
P_5	010000
P_6	010001
P_7	0101
P_8	0111
P_9	11

prefixes	nexthop
010000/6	P_5
010001/6	P_6
0000/4	P_1
0001/4	P_2
0101/4	P_7
0111/4	P_8
001/3	P_3
01/2	P_4
11/2	P_9
*/0	default

Fig. 1 *Representation of the routing prefixes with prefix trie*

list the detailed subtrie construction algorithm below. The time complexity is $O(2^H NW)$ since the cost for constructing subtrie array is 2^H and each node will be accessed once.

```
1. Subtree construction algorithm:
2. Input: The root of the constructed binary tree from the routing table
3. Output: The constructed subtrees and the corresponding bit stream
4. Constructor (Current_Root, Current_Depth) BEGIN
5.     If ((the depth of the deepest child in the subtree - Current_Depth) = = H)
6.         Generate_Subtree (Current_Root);
7.     Else
8.         Constructor (Current_Root- > Left_Child, Current_Depth + 1);
9.         Constructor (Current_Root- > Right_Child, Current_Depth + 1);
10.        If ((the length of the longest unprocessed prefix - Current_Depth) = = H)
11.            Generate_Subtree (Parent, Current_Root);
12.    Endif
13. END
```

By using the subtrie construction algorithm, we can divide the binary trie into multiple subtries. Let the height of the subtries equal two, the binary trie can be divided into four subtries, as shown in Fig. 2*a*. Each routing prefix will be assigned to at least one of the subtries. Next, the four bit streams, which correspond to the subtries' roots, will be inserted into the TCAM as a substitution for the original prefixes. As a consequence, for each representative prefix, there is a corresponding entry in the intermediate table with

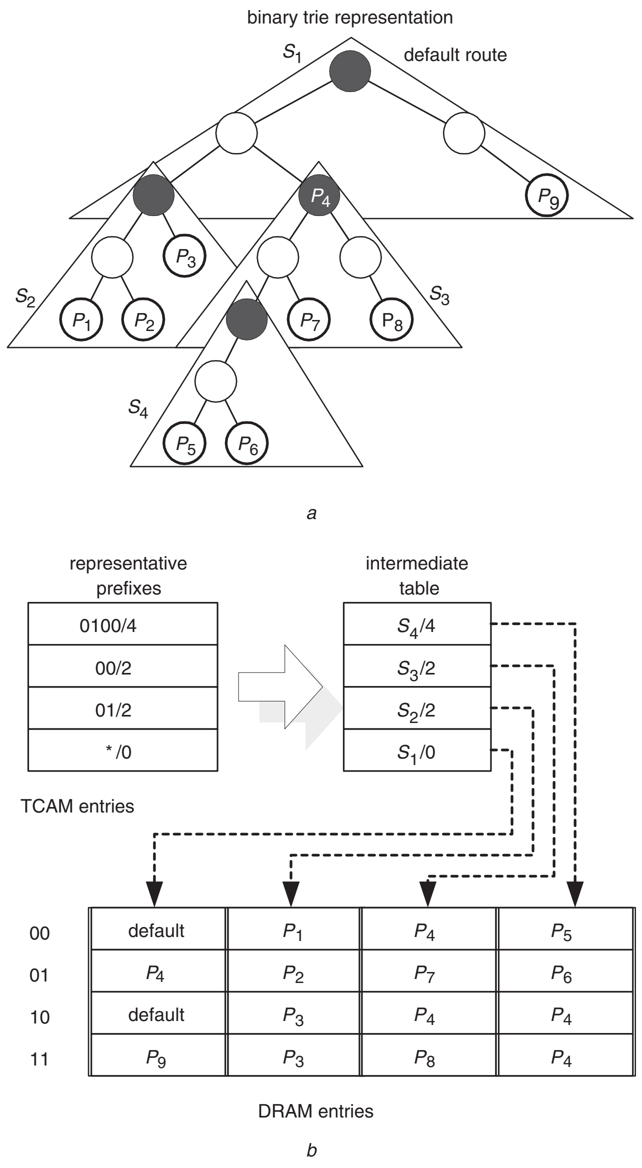


Fig. 2 *Reorganisation of the binary trie into four subtries whose heights are two*

the subtrie information. Then, we expand each subtrie completely and put it into DRAM. In our example, each subtrie will be expanded to four ($=2^2$) entries.

3.1 Routing lookup

The routing lookup procedure consists of two steps. The first step is to find out the best matching '**subtrie**' while the second step extracts the best match prefix based on the intermediate table. In the first step, the TCAM lookup result indicates the corresponding entries in the intermediate table, as shown in Fig. 2*b*. Each entry shows the address of the subtrie and its depth. Since the height of the subtries is equal to two, the following two bits behind the effective bit length (subtrie depth) are used to select the corresponding entry. For example, we perform the search for address 000111. The third entry will be selected in the TCAM lookup and the corresponding subtrie (S_2) and depth (i.e. the effective bit length: 2) are shown in the third entry of the intermediate table. Then the third and fourth bits ('0' and '1', respectively) will be used to indicate the second entry in the S_2 and derive the result of BMP 'P_2'.

In the proposed scheme, each routing lookup needs one TCAM and two DRAM accesses, which is one DRAM access more than the original implementation. To prevent any performance degradation, the use of hardware pipelining design is necessary. Although pipelining design will increase the circuit design complexity, it has been widely adopted in the design of network processors due to the advance of VLSI technology [11]. With pipelining, the proposed scheme can be accomplished as efficiently as one lookup per clock cycle.

3.2 Route update

Since the original routing prefixes have been encapsulated in the subtries, each route update must enquire the best matching subtrie to check whether the updated prefix can be located. If yes, the related entries in the DRAM will be modified. Otherwise, the routing prefix is inserted into the TCAM directly. After accumulating a certain number of such entries, we can rebuild the subtries to keep the number of TCAM entries low. The worst-case update time is $maximum(W/2,\ 2^{height\ of\ subtrie})$.

3.3 Redundant route remove

Although the aggregation of redundant routes for a single-site or campus-level network is straightforward, it is considerably more difficult to aggregate routes at a larger scale, including going across multiple backbone providers. Moreover, it requires close co-operation among service providers. The main reason for poor aggregation is the increasing trend toward end-sites-connectivity multiple service providers. In our experiments, we found that there are many such routes in the routing tables. We could remove these redundant routes without affecting the packet forwarding.

We start the redundant route remove algorithm with the root of the prefix trie and default route. When we reach a node with prefix, we would compare whether its nexthop value is identical to that of the ancestor prefix. If yes, we identify this prefix as redundant route and drop it. Otherwise, we use the nexthop value to examine the successive nodes. The time complexity is $O(NW)$.

```
1. Redundant route remove algorithm:
2. Input: The root of the binary trie which is constructed from the routing table.
3. Output: The routing table after removing redundant routes.
4. Remove (Current_Node,Nexthop) BEGIN
5. If (Current_Node corresponds to a routing prefix)
6. If (Current_Node−>Nexthop == Nexthop)
7. RemoveRoute(Current_Node−>Route);
8. Else
9. Nexthop = Current_Node−>Nexthop;
10. Remove (Current_Node−>Left_Child, Nexthop);
11. Remove (Current_Node−>Right_Child, Nexthop);
12. Endif
13. END
```

4 Performance evaluation

To investigate the scalability of the proposed scheme, we use the real data available from the IPMA [12] and NLANR [13] projects for comparison. These data provide a daily snapshot of the routing tables used by some major network access points (NAPs). We set the height of the subtrie as four and eight to investigate the effects of different subtrie height. The performance metrics include the number of generated TCAM entries and the required

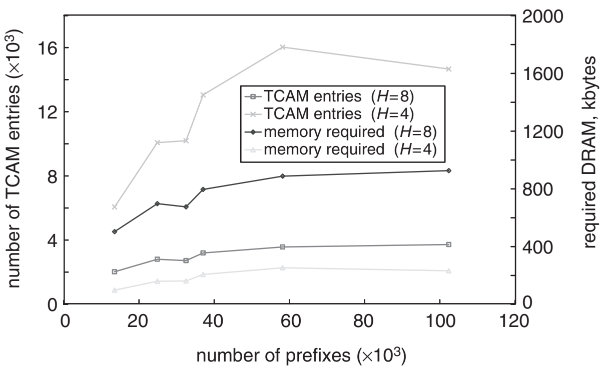


Fig. 3 *Performance metrics for different routing tables*
Subtrie height = 4.8

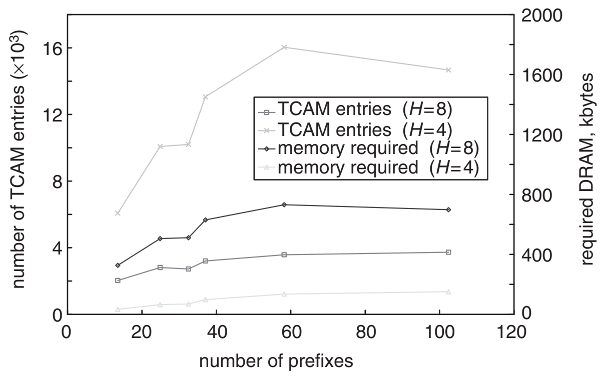


Fig. 4 *Performance metrics for different routing tables*
Subtrie height = 4.8; dynamic

DRAM storage. We also list the number of different prefix lengths to examine the update cost. Through experiments, we demonstrate that the proposed scheme features much fewer TCAM entries with less extra DRAM.

Figure 3 shows the processing results for different routing tables. Both performance metrics are proportionate to the number of prefixes. For the largest table with 102 271 prefixes (NLANR), it generates 15 998 and 3691 TCAM entries accompanying 249 kbytes and 922 kbytes DRAM, respectively. It is straightforward that a larger subtrie could reduce the number of TCAM entries, but also results in more required DRAM. By changing the height of the subtrie, the TCAM entries and the required DRAM can be adjusted according to practical environments. Note that the different characteristics of the routing tables (e.g. the number of prefixes and nexthops) might cause variation in the numerical results.

In addition to the basic scheme, we could adopt various heuristics to improve the performance. Accordingly, we show the performance of two enhancements. In the first enhancement, we adopt a dynamic mechanism based on the number of prefixes in the subtrie to decide how to implement the subtrie. This is based on the observation in the experiments. We noticed that some subtries only occupied one prefix. In such a case, we do not have to allocate DRAM for this representative prefix. Only the dense prefixes will be merged. This enhancement will eliminate the required DRAM, but will not affect the number of TCAM entries, as shown in Fig. 4. The decreased DRAM space varies from 15% to 35%.

The second enhancement merges prefixes before executing the subtrie construction algorithm. This enhancement is based on the observation that there is a large number of single-path prefixes in the routing tables [5]. The sparseness

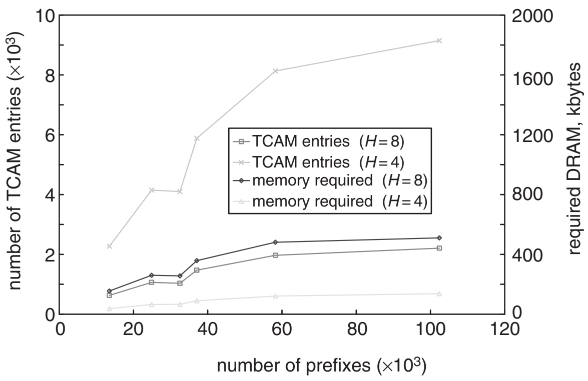


Fig. 5 *Performance metrics for different routing tables*
Subtrie height = 4.8; dynamic; path compression

Table 1: Number of difference prefix lengths

NAPs	Number of prefixes	Number of different lengths		
		Original	$H=4$	$H=8$
Paix	13 395	20	20	18
AADS	25 407	23	22	15
PacBell	32 388	23	22	18
Mae-West	36 943	23	23	19
Mae-East	58 101	23	23	20
NLANR	102 271	25	23	18

caused by the single-path prefixes will make the subtrie construction inefficient. We can apply limited path compression to eliminate the single-path leaves before the subtrie construction. When we reach a node with a prefix in the prefix trie and there is only one longer prefix in the children nodes, we could merge the longer prefixes into a shorter one, and the resultant prefix includes the information of the merged prefix and a pointer to the original one. This is a special case of the single-path prefix. While the resultant prefix is fetched in the lookup, the bit stream of the longer prefix is compared. If it is matched, the nexthop of the longer prefix is used. Otherwise, the nexthop of the shorter prefix is preferred. Hence in the worst case, the lookup procedure requires one extra memory access. As shown in Fig. 5, we are able to reduce both the required TCAM entries and DRAM significantly. A 100 K-entry routing table could be composed of 2187 TCAM entries and 546 kbytes DRAM (subtrie height = 8). Only about 30–60% TCAM entries of the basic scheme is required in the process, yet is accomplished at a slower speed.

Finally, we present the number of different prefix lengths to estimate the update cost in Table 1. The update cost could benefit from less distinct lengths. Since the TCAM update cost is low, the update performance ties to the subtrie update in DRAM. We could update the modified subtrie into different locations followed by changing pointers to minimise forwarding suspension.

5 Conclusions

This study investigates the related issues in TCAM-based forwarding engine design. To make use of the TCAM in IPv6 routing lookup, we need a more efficient approach to fulfill the requirements of the forwarding engine as well as to improve the routing lookup performance. The proposed algorithm reduces the number of TCAM entries by merging routing prefixes into subtries according to the associative positions. The subtries' roots and their interior information are stored in the TCAM and DRAM, respectively. Each routing lookup procedure consists of one TCAM and DRAM access that can proceed in pipelining. By adjusting the parameters of the subtries, including the height and dynamic property, we can decide the number of generated TCAM entries and the required DRAM size. Furthermore, we adopt the technique of path compression to reduce both the required TCAM and DRAM. The enhancement will incur one extra memory access due to the possible 'incorrect' match. In the experiments, we have illustrated various performance metrics with different settings. The proposed algorithm can eliminate 98% of TCAM entries using only 550 kbytes DRAM in the best case. Although the complex IPv6 routing tables are not yet available, we believe that this scheme would simplify the design of the IPv6 routers by alleviating the TCAM cost dramatically.

6 References

1 Rekhter, Y., Li, T.: 'An architecture for IP address allocation with CIDR', RFC 1518, Sept. 1993

2 Degermark, M., Brodnik, A., Carlsson, S., and Pink, S.: 'Small forwarding tables for fast routing lookups'. Proc. ACM SIGCOMM' 97, Cannes, France, Sept. 1997, pp. 3–14

3 Gupta, P., Lin, S., and McKeown, N.: 'Routing lookups in hardware at memory access speeds'. Proc. IEEE INFOCOM'98, San Francisco, CA, USA, March 1998

4 Wang, P.-C., Chan, C.-T., and Chen, Y.-C.: 'A fast IP lookup scheme for high-speed networks', *IEEE Commun. Lett.*, 2001, **5**, (3), pp. 125–127

5 Nilsson, S., and Karlsson, G.: 'IP-address lookup using LC-tries', *IEEE J. Sel. Areas Commun.*, 1999, **17**, (6), pp. 1083–1029

6 Waldvogel, M., Varghese, G., Turner, J., and Plattner, B.: 'Scalable high speed IP routing lookups'. Proc. ACM SIGCOMM '97, Cannes, France, Sept. 1997, pp. 25–36

7 Lampson, B., Srinivasan, V., and Varghese, G.: 'IP lookups using multiway and multicolumn search', *IEEE/ACM Trans. Netw.*, 1999, **7**, (4), pp. 324–334

8 Shah, D., and Gupta, P.: 'Fast updating algorithms for TCAMs', *IEEE Micro*, 2001, **21**, (1), pp. 36–47

9 Srinivasan, V., and Varghese, G.: 'Fast IP lookups using controlled prefix expansion', *ACM Trans. Comput.*, 1999, **17**, (1), pp. 1–40

10 Wang, P.-C., Chan, C.-T., and Chen, Y.-C.: 'Performance enhancement of IP forwarding by reducing routing table construction time', *IEEE Commun. Lett.*, 2001, **5**, (5), pp. 230–232

11 Haas, R., Kencl, L., Kind, A., Metzler, B., Pletka, R., Waldvogel, M., Frelechoux, L., Droz, P., and Jeffries, C.: 'Creating advanced functions on network processors: experience and perspectives', *IEEE Netw.*, 2003, **17**, (4), pp. 46–54

12 Merit Networks, Inc. Internet performance measurement and analysis (IPMA) statistics and daily reports. See http://www.merit.edu/ipma/routingtable/

13 NLANR Project. See http://moat.nlanr.net/