This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE Globecom 2011 proceedings.

# Parallel Name Lookup for Named Data Networking

Yi Wang, Huichen Dai, Junchen Jiang, Keqiang He, Wei Meng, Bin Liu
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, 100084, China

***Abstract*—Name-based route lookup is a key function for Named Data Networking (NDN). The NDN names are hierarchical and have variable and unbounded lengths, which are much longer than IPv4/6 address, making fast name lookup a challenging issue. In this paper, we propose a parallel architecture for NDN name lookup called Parallel Name Lookup (PNL) which leverages hardware parallelism to achieve high lookup speedup while keeping a low and controllable memory redundancy. The core of PNL is an allocation algorithm that maps the logically tree-based structure to physically parallel modules, with low computational complexity. We evaluate the PNL's performance and show that PNL dramatically accelerates the name lookup process. Furthermore, with certain knowledge of prior probability, the speedup can be significantly improved.**

## I. Introduction

Named Data Networking [1] is proposed recently as the clean-slate network architecture for future Internet, which no longer concentrates on "where" the information is located, but "what" the information (content) is needed. NDN uses names to identify every piece of contents instead of IP addresses for hardware devices attached to IP network.

NDN forwards packets by names, which implies a substantial re-engineering of forwarding and its lookup mechanism. We highlight the challenges as below:

1) Unbounded processing time and low throughput. NDN names, unlike fixed-length IP addresses, may have variable lengths and no externally imposed upper bound. This makes name lookup at line-speed extremely challenging as arbitrarily long name will cost a lookup time linear to its length.
2) Inefficient prefix aggregation. NDN names, unlike classless IP addresses, have hierarchical structure and coarser granularity, consisting of a series of delimited components. NDN's longest prefix matching differs from that of IP in the way that NDN must match a prefix at the end of a component, rather than at any digit in IP. Therefore, traditional prefix aggregation of NDN will be far less efficient in NDN name lookup.
3) Higher update rate. NDN name lookup is mixed with more frequent updates than Forwarding Information Base (FIB) in IP router. Because, besides routing refreshing, NDN name table update is also needed every time new content is stored in local memory as well as when old one is replaced, which is extremely dynamic. As this information is designed to be stored together with FIB in NDN router, the name lookup must support fast insertion and deletion with reasonably low overhead.

In this paper, we propose, analyze and implement Parallel Name Lookup (PNL), a fast name lookup architecture using parallel memory resources to obtain high speedup for NDN. PNL is based on an analogue of IP prefix tree, called Name Prefix Tree (NPT) where each node represents a set of components. The key observation is that millions of nodes of NPT can be selectively grouped and allocated into parallel memories. When any input name comes, it will traverse and be matched in some memories so that when multiple input names come concurrently, almost all memories will be visited and work simultaneously, achieving a high overall speedup. Indeed, the allocation requires some nodes to be duplicated, causing potential memory explosion. But we show that NPL fully exploits parallelism with strictly bounded redundancy.

The application of PNL is beyond NDN name lookup as the methods used are independent to how name or component is written. We develop NPL to be a generic architecture to accelerate name lookup as long as such name causes any of the challenges listed above. Especially, we make the following contributions:

1) We propose PNL that performs accelerated NDN name lookup and give worst case analysis. The core of PNL is an allocation algorithm that allocates the logically tree-based structure to parallel physical modules with low and controlled storage redundancy. To handle highly frequent update, the allocation algorithm is designed to be performed with small computational complexity.
2) We implement and test the prototype, the evaluation results indicate that PNL could achieve remarkable average speedup for name lookup. The speedup significantly augments as the number of modules k increases.

The rest of this paper is organized as follows. Section II introduces NDN, NDN naming strategy and the name prefix lookup problems caused by NDN names. To solve the problem, Section III proposes a promising solution and Section IV describes the PNL algorithm in detail. Section VI is the related work and we conclude our research in Section VIII.

## II. Name Lookup of NDN

### A. NDN Background

Despite the novelty of NDN, NDN operations can be grounded in current practice, routing and forwarding of NDN are semantically the same as that of IP network. What differs is that, every piece of content in NDN is assigned a name, and NDN routes and forwards packets by these names, rather than IP addresses.

978-1-4244-9268-8/11/$26.00 ©2011 IEEE

Data requesters initiate communications in NDN. A data requester sends out an Interest packet with the requested content name, routers forward the Interest by looking up its name in the FIB, when the Interest reaches a device that provides the required data, a Data packet is sent back to the requester.

### B. Naming in NDN

An NDN name is hierarchically structured and composed of explicitly delimited components, while the delimiters, usually slashes or dots, are not part of the name. The hierarchical structure, like that of IP address, enables name aggregation and allows fast name lookup by longest prefix match, and aggregation in turn is essential to the scalability of routing and forwarding systems. Though NDN routes and forwards packets by names, names are opaque to the network, which means routers are not aware of what a name means, but only know the boundaries between components.

The naming strategy may include designing name structure, name discovery and namespace navigation schemes, and is not limited to one solution but now an open research question. For the purpose of early exploring the name-based route lookup mechanism before the naming specification finally goes to standard, in this paper, we temporarily use hierarchically reversed domains as NDN names, and apply our proposed PNL scheme instead. For example, *maps.google.com* is hierarchically reversed to *com/google/maps*, and *com*, *google*, *maps* are three components of the name.

## III. Framework of Parallel Name Lookup

### A. Name Prefix Tree (NPT)

NDN names are composed of explicitly delimited components, hence they can be represented by NPT. NPT is of component granularity, rather than character or bit granularity, since the longest name prefix lookup of NDN names can only match a complete component at once, i.e., no match happens in the middle of a component. Fig. 1 shows seven names which can be used to construct a NPT illustrated in Fig. 2. Each edge of NPT stands for a name component and each node stands for a lookup state[1]. Name prefix lookups always begin at the root.

When the lookup for a given name starts, it firstly checks if first component matches one of the edges originated from the root, i.e., the level-1 edge. If so, the transfer condition holds and then the lookup state transfers from the root to the pointed level-2 node. The subsequent lookup process proceeds iteratively. When the transfer condition fails to hold or the lookup state reaches one of the leaf nodes, the lookup process terminates and outputs the ports' number that the last state corresponds to. For example, in Fig. 2, given name */com/google/maps*, the first component *com* matches a level-1 edge and the lookup state transfers from root (state 1) to state 2; the level-2 transfer condition also holds and the lookup state continues to transfer to state 5; the third name component *maps* does not satisfy the level-3 transfer condition and the lookup process terminates, with the lookup state remaining to be state 5.

NPT possesses faster lookup speed and less storage overhead when compared to the traditional character tree. However the lookup process still needs to search from the root to lower level nodes in serial, without taking full advantage of the router hardware parallelism.

### B. Motivation of Parallel Name Lookup

We group up the states of different levels in the NPT and store the groups to individual physical modules, one group corresponding to one physical module. As Fig. 3 depicts, 15 states are allocated into 3 groups, regardless of their levels, and stored into 3 physical modules, while the states maintain the same transfer relations (or topology) as that in the NPT.

Below we show how our new scheme works. Fig. 4 illustrates a scenario of 3 physical modules running in parallel. Look at the second line and suppose that we are at state 6 in module 3. After a match of *sina*, we want to transfer to state B in module 3. Then the lookup procedure examines if there exist conflicts (explained in Section IV). If not, the lookup state transfers to state B and continues to explore next state transition. When multiple states intend to transfer to states in the same module at the same time, conflict arises, which we show by dotted rectangles in Fig. 4. Then only one state transfers successfully, while others have to wait for the second chance to preemptive resource. In Fig. 4, after the match of *yahoo*, the first and third line both intend to transfer to states (4 and 1) in module 1, then the state of the first line succeeds in transferring and the third line remains waiting. At the same time, module 3 stays idle since no state transfers into it. Consequently, conflicts result in state transfer delay and physical module being idle. Therefore, the less conflict, the better performance.

Fortunately, we use an effective strategy to distribute states into modules. The framework is following. First, we calculate the access probability of each state based on prior knowledge of names' access probability, if name has no prior access probability, we give a default value. Second, we set up a threshold $P_t$. If there is a state whose access probability $P$ is lager than $P_t$, make $m = \lceil \frac{P}{P_t} \rceil - 1$ duplications of that state, each with access probability $P/m$. Then allocate all the states, both originals and duplications, into $k$ physical modules, and the sum of the access probabilities of states assigned to one module is the total access probability of that module. Our purpose is that access probability deviation between physical modules does not exceed $P_t$. As Fig. 3 depicts, we duplicate state 1 twice and state 2 once, the duplications or copies are shown by dotted circles. Fig. 5 shows the matching process for the same name components in Fig. 4, our strategy effectively reduces conflicts from 6 times to only 1 time.

## IV. Algorithm Design and Analysis

### A. PNL Algorithm

Section III has provided the general framework of our PNL method, in this section we continue to explore it in detail.

[1]In the rest of this paper, if not confused, we use *state* and *node* interchangeably.

/com/yahoo/news
/com/yahoo/music/new
/com/google/news
/com/google
/cn/com/sina/news
/cn/com/sina/mail
/cn/yahoo/news

Fig. 1. Examples of NDN names.

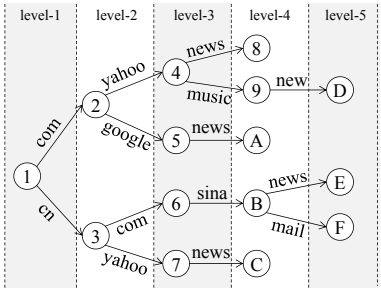


Fig. 2. An example of NPT.

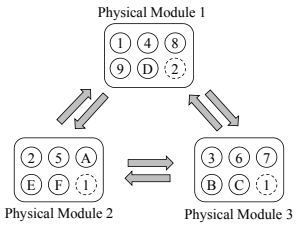


Fig. 3. Allocate 15 states into 3 groups, stored into 3 physical modules. Dotted circles are duplications.

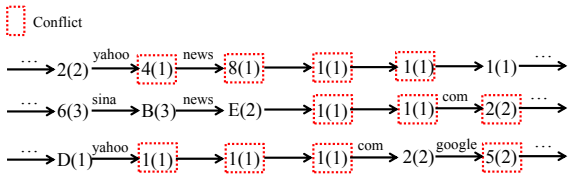


Fig. 4. Parallel prefix lookup without state duplications, more conflicts.

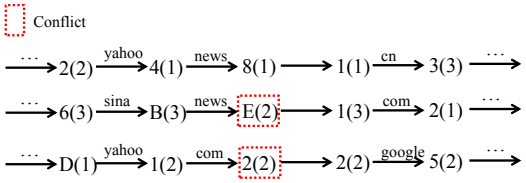


Fig. 5. Parallel prefix lookup with state duplications, less conflicts.

The Parallel Name Lookup algorithm consists of three sub-procedures:

1) Build NPT based on the names stored in FIB;
2) Calculate the access probability of every node in an NPT according to the prior knowledge of each domains access probability. If unknown, set the access probability of all the names to the same average value;
3) Allocate all the states, including originals and duplications, to $k$ physical modules.

Procedure 2 employs Algorithm 1 to calculate the access probabilities of all the nodes in $O(N)$ time. Algorithm 1 looks up every name in the PNT, the key step is adding each name's access probability to the nodes on the traversing path, and then normalizing all nodes' access probabilities at the end of the procedure. The reason why we can do like this is that, we think the access probability of each node is "passed on" to it by its predecessors, and there is no back edge in NPT, thus a node's access probability is just the normalized accumulation of all the probabilities passed to it by its predecessors.

After Procedure 2, procedure 3 allocates all the states into $k$ physical modules, sums up the access probabilities of all the states assigned to a physical module as that physical module's access probability, guaranteeing that access probability deviation between physical modules is less than a threshold $P_t$. Algorithm 2 is designed to achieve this target, line 2 sorts all the nodes by their access probabilities, then each physical module $Q_i$ is initialized as NULL. Line 5 to 8 calculate $t$, the times that a state will be copied ($t \leq k$), and the access probability of each copied state drops to $1/t$ of the original. Line 9 to 12 is the loop that copies states to modules and recalculate the modules' access probabilities. At last $k$ physical modules with the access probability deviation less than $P_t$ are returned.

Below we prove that algorithm 2 guarantees the access probability deviation between physical modules is less than $P_t$ by mathematical induction.

*Proof:* (1) Before the loop of line 9 to 12 begins, the access probabilities of all the modules are 0, hence, the differences between them are also 0, and $0 < P_t$; (2) Assume that at $n$-th iteration beginning, the access probabilities of $k$ modules satisfy $P(Q_1) < \cdots < P(Q_k)$, and $|P(Q_x) - P(Q_y)| < P_t, 1 \leq x, y \leq k$; (3) After Running from line 10 to line 11 (one iteration), $P(Q_1^{'}) = P(Q_1) + \frac{P(S_i)}{t}$ and $P(Q_x^{'}) = P(Q_x), 2 \leq x \leq k$. For $2 \leq x, y \leq k$, it still keeps $|P(Q_x^{'}) - P(Q_y^{'})| < P_t$. For $|P(Q_1^{'}) - P(Q_x^{'})| = |P(Q_1) + \frac{P(S_i)}{t} - P(Q_x)| = |\frac{P(S_i)}{t} - (P(Q_x - P(Q_1))|$. Because $\frac{P(S_i)}{t} < P_t$ and $0 \leq P(Q_x - P(Q_1)) \leq P_t$, $\Rightarrow |\frac{P(S_i)}{t} - (P(Q_x) - P(Q_1))| < P_t$. Finally, at $(n+1)$-th iteration beginning, it still keeps $|P(Q_x^{'}) - P(Q_y^{'})| < P_t$, so it proves our algorithm is correct. ■

The order of running time of algorithm 2 is $O(N * \log(N) + N * k^2 \log(k))$, due to $k << N$, we omit the $N * k^2 \log(k)$ part and arrive at $O(N * \log(N))$. So, the total time complexity of building PNL is: $O(N) + O(N) + O(N * \log(N)) = O(N * \log(N))$.

**Algorithm 1** Node Access Probability Calculator (NAPC)

```
1:  procedure NAPC(PNT, d_1, d_2, ..., d_N)
2:      P(ALL) ← 0
3:      for i ← 1 to N do            ▷ N is the number of nodes
4:          node ←root(PNT)
5:          (d_i1, d_i2, ..., d_ik) ←split(d_i)
6:          for j ← 1 to k do
7:              P(node) ← P(node) + P(d_i)
8:              node ←NextState(node, d_ij)
9:              P(ALL) ← P(ALL) + P(d_i)
10:         end for
11:         for each node ∈ PNT do
12:             P(node) ← P(node)/P(ALL)
13:         end for
14:     end for
15: end procedure
```

**Algorithm 2** Allocate States to Modules (ASM)

1: **procedure** ASM($S_1, S_2, \cdots, S_N$,$k$,$P_t$)
2: $(S_1^{'}, S_2^{'}, \cdots, S_N^{'})$←SORT$(P(S_1), P(S_2), \cdots, P(S_N))$
3: $(Q_1, Q_2, \cdots, Q_k)$←$(\emptyset, \emptyset, \cdots, \emptyset)$
4: **for** $i \leftarrow N$ to 1 **do**
5: $t \leftarrow \lceil \frac{P(S_i^{'})}{P_t} \rceil$ ▷ $\frac{P(S_i^{'})}{t} < P_t$
6: **if** $t > k$ **then**
7: $t = k$ ▷ at worst copy $S_i^{'}$ to all $k$ modules
8: **end if**
9: **for** $j \leftarrow 1$ to $t$ **do**
10: $Q^{'}$ ←MIN$(P(Q_1), P(Q_2), \cdots, P(Q_k))$
11: $Q^{'} \leftarrow Q^{'} \cup S_i^{'}, P(Q^{'}) \leftarrow P(Q^{'} + \frac{P(S_i^{'})}{t})$
12: **end for**
13: **end for**
14: **return** $Q_1, Q_2, \cdots, Q_k$ ▷ $|Q_i - Q_j| < P_t, 1 < i, j < k$
15: **end procedure**

### B. Fast Update Support

PNL can support fast name update by enduring a negligible unbalance. When a new name is added to NPT, it probes itself from root node of NPT to leaf node step by step. If all nodes on the path exist, it only change these nodes' access probabilities. Otherwise, the name adds new nodes to the NPT, and uses algorithm 2 (line 5 to 11) to add new nodes to physical modules. Compare with more than 1 million names in NPT, and a new name probably has very low access probability and caused a negligible influence on the global balance of PNL modules' access probability. It takes $O(l + k * \log(k))$ ($l$ is the number of components in the name), i.e., $O(k*\log(k))$ time complexity to update one name to PNL.

### C. The Worst Cases Analysis

In the worst case, there is no prior access probability of states. Suppose each state's probability is less than $P_t$, so states are allocated to modules without duplications, one state only stored in one module. The head state of one queue can only go to one module for processing. When $q = k$ ($q$ denotes number of queues, every module is assigned exactly one queue), $k$ head states need to be distributed to $k$ physical modules at the same time, it is equivalent to *Input Queued Switch* with the problem of head-of-line (HOL) blocking. It is proved that under uniform traffic arrival, the throughput of an input buffered switch is at most $2 - \sqrt{2} = 58.6\%$ [2], the total speedup of $q$ queues can arrive at $(2 - \sqrt{2})k$. In the extreme situation, all head states of $q$ queues go to the same physical module, only one module works. Hence, in the worst case, PNL has 1 to $(2 - \sqrt{2})k$ speedup.

To bound the extra memory of state duplication, we denote $a_i$ as the number of state $S_i$ with $a_i = \lceil \frac{P(S_i)}{P_t} \rceil$. After allocation, the total number of states $N^{'} = \sum_{i=1}^{N} a_i = \sum_{i=1}^{N} \lceil \frac{P(S_i)}{P_t} \rceil < \sum_{i=1}^{N} (\frac{P(S_i)}{P_t} + 1) < \sum_{i=1}^{N} \frac{P(S_i)}{Pt} + N$, so the number of new adding states $\Delta = N^{'} - N < \sum_{i=1}^{N} \frac{P(S_i)}{P_t} = \frac{1}{P_t}$. Suppose the maximal memory requirement of one state is $M$, so the upper bound of total extra memory is $\frac{M}{P_t}$ in the worst case.

## V. Evaluation

In this section, we utilize the domain information from the DMOZ [3] and ALEXA [4] to conduct two simulations, respectively. Table I shows the number of level-1 to level-9 domains obtained from these two websites. We *hierarchically* reverse all the domains as NDN names, and then build an NPT on DMOZ and ALEXA, respectively. The number of states at different tree levels of the NPT are reflected in Table II.

TABLE I
Number of Level-1 to Level-9 Domains.

| Domains level | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| DMOZ | 0 | 1179492 | 246082 | 20261 | 4659 | 400 | 34 | 15 | 3 |
| ALEXA | 0 | 875665 | 119154 | 817 | 69 | 54 | 19 | 4 | 0 |

TABLE II
States at Different NPT Levels.

| NPT level | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| DMOZ | 1 | 306 | 1183335 | 249444 | 22022 | 5042 | 447 | 52 | 18 | 3 |
| ALEXA | 1 | 252 | 871349 | 117952 | 544 | 129 | 75 | 21 | 4 | 0 |

**Simulation 1** *(Without prior access probability of names)*: This simulation is based on the data obtained from DMOZ. We can infer from Table I that DMOZ provides 1450946 domains in total, without knowing the access probability to each of them. So we first assume that each domains has the same access probability. The access probability of a node in NPT is determined by the matching ratio of its preceding edge, which, under this hypothesis, is exactly the appearing frequency of the edge's corresponding component of all components. The access probability Cumulative Distribution Function (CDF) of the nodes in this simulation is shown in Fig. 6. Then we divide all the 1450946 names evenly into 10 groups as input to PNL and get simulation results.

**Simulation 2** *(With prior access probability of names)*: We conduct this simulation based on the page visits of 1 million domains retrieved from ALEXA. We take the three-month average percent of global pageviews on every domains as its prior access probability. According to every domain's access probability, we utilize algorithm 1 to calculate the access probability of every node in the NPT. Fig. 6 also shows the access probability CDF of the nodes in this simulation.

Next, we duplicate some nodes based on their access probabilities and allocate all the nodes into 10 groups, then the PNL algorithm is invoked and simulation results are output.

**Simulation results**: As is shown in Fig. 7, when $P_t = 0.05$ and each module has 3 queues, the PNL approach dramatically accelerates the name lookup process, and the speedup

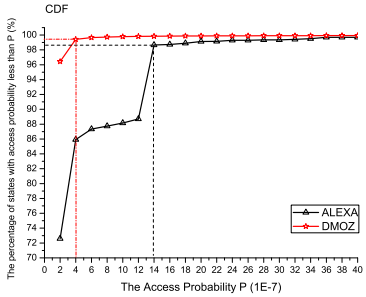


Fig. 6. The cumulative percentage of states with probability less than $P$.

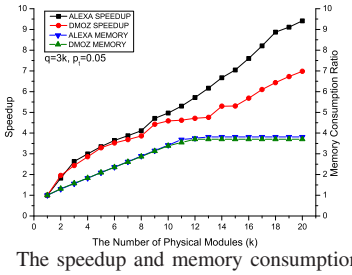


Fig. 7. The speedup and memory consumption ratio over number of physical modules $k$.

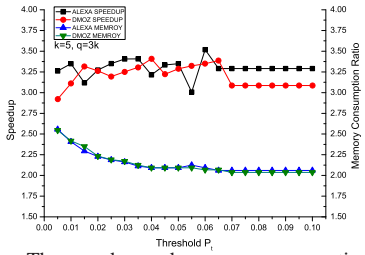


Fig. 8. The speedup and memory consumption ration over $P_t$.

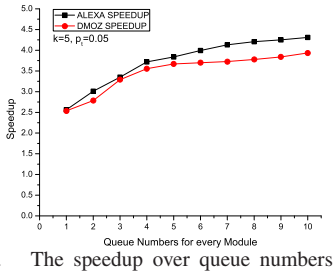


Fig. 9. The speedup over queue numbers ($q$) for every module.

remarkably augments as the number of modules $k$ increases. By comparing the speedups corresponding to ALEXA and DMOZ, we arrive at the conclusion that with certain knowledge of prior probability, the speedup can be significantly improved, the more physical modules, the larger the margin. When $k = 20$, the speedup corresponding to ALEXA and DMOZ amazingly reaches 9.5 and 7 respectively. Fig. 7 also reveals that as the number of physical modules $k$ increases, ALEXA and DMOZ both exhibit the demand of more memory. But when $k > \lceil \frac{P(S_i)}{P_t} \rceil$ ($\lceil \frac{P(S_i)}{P_t} \rceil = 13$ in this experiment), which means the modules are more than state duplications, the memory consumption remains the same.

Fig. 8 shows that as $P_t$ increases, the speedup may vary within a range but with no explicit discipline. When $P_t$ exceeds a threshold, the speedups of DMOZ and ALEXA will not change any more. Also, with the increase of $P_t$, the states needs to be copied become less, consequently the memory overhead decreases. And Fig. 9 tells that when the number of queues associated with each physical module increases, it removes the head-of-line blocking problem when only one queue is assigned to a physical module, thus the speedup is improved again.

## VI. Related Work

Many research works have been done in the traditional IP Prefix Lookup, and some of them are helpful for us to design and realize fast Name Prefix Lookup in NDN. In [5], the authors propose a parallel IP lookup scheme by duplicating FIB to different TCAM modules. The previous work on DPPC-RE [5] distributes the IP prefix to TCAMs on the basis of prefixes' access probabilities [6], and it improves the lookup performance. Z. Genova *et al.* [7] hash URLs to the fixed-length signatures, and lookup the signature in the hash table, which also works well. However, these methods consider URLs as indivisible entities, thus cannot provide prefix matching capability. Some other methods try to decompose a URL into components and build a components tree to aggregate URL [8]. The URL components tree is similar to our Name Prefix Tree, which can only offer basic prefix lookup. Shue *et al.* [9] compare the performance of Name-based forwarding with Prefix-based forwarding, which shows name lookup is feasible, but slower than prefix lookup in the traditional prefix tree. J. Jiang *et al.* [10] is proposed to parallelize regular expression matching in intrusion detect system, and it only thinks about simple state allocation. We need a much faster name lookup mechanism to server NDN forwarding lookup.

## VII. Acknowledgment

This paper is supported by NSFC (60873250, 61073171), 973 project (2007CB310702), and Tsinghua University Initiative Scientific Research Program and the Specialized Research Fund for the Doctoral Program of Higher Education of China (20100002110051).

## VIII. Conclusion

We present an NDN lookup method named PNL which achieves a high lookup speedup by utilizing the physical parallelism. Based on the characteristics and the prior access probability of names, an algorithm with $O(N * \log(N))$ complexity is proposed to build NPT, duplicate and allocate nodes to physical modules, the algorithm also guarantees that the access probability deviation between physical modules is less than a threshold. We design a hardware prototype for PNL, and evaluate its performance by two domain sets. The simulation results show that using PNL, the speedup remarkably augments as the number of modules increases. The memory overhead first grows up at a low speed, then remains at a controllable level. Going on to reduce memory overhead is our future work.

## References

[1] L. Zhang, D. Estrin, V. Jacobson, and B. Zhang, "Named data networking (ndn) project," in *Technical Report, NDN-0001*, 2010.

[2] M. J. Karol, M. Hluchyj, and S. Morgan, "Input vs. output queuing on a space-division packet switch," *IEEE Transactions on Communications*, vol. 35, pp. 1347–1356, 1987.

[3] ODP - Open Directory Project. [Online]. Available: http://www.dmoz.org/

[4] Alexa the Web Information Company. [Online]. Available: http://www.alexa.com/

[5] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, "Dppc-re: TCAM-based distributed parallel packet classification with range encoding," *IEEE Transactions on Computers*, vol. 55, no. 8, pp. 947–961, 2006.

[6] D. Lin and B. liu, "Route table partitioning and load balancing for parallel searching with tcams," in *21st IEEE International Parallel & Distributed Processing Symposium - IPDPS*, 2007.

[7] G. Z. and C. K., "Managing routing tables for url routers in content distribution networks," *International Journal of Network Management*, vol. 14, pp. 177–192, 2004.

[8] B. Michel, K. Nikoloudakis, P. Reiher, and L. Zhang, "Url forwarding and compression in adaptive web caching," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2000, pp. 670 –678 vol.2.

[9] C. Shue and M. Gupta, "Packet forwarding: Name-based vs. prefix-based," in *IEEE Global Internet Symposium, 2007*, May 2007, pp. 73 –78.

[10] J. Jiang, X. Wang, K. He, and B. Liu, "Parallel architecture for high throughput dfa-based deep packet inspection," in *Communications (ICC), 2010 IEEE International Conference on*, May 2010, pp. 1 –5.