ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 4, Issue 12, December 2017

DAAM: Direct Address Accessing Method for Scalable Forwarding in NDN

[1]Anjali Kumari, [2]Anurudh Kumar, [3] Ms Devikala K, [4]R.P. Seenivasan, [5]Dr K.Suresh Joseph
[1] M.Tech (NIE) ,[2,][3] Ph.D Research Scholar,[4][5]Assistant Professor Pondicherry University

***Abstract* - Named data Networking (NDN) is emerging internet architecture according to today scenario about devices which function in internet of everything environment. NDN-architecture forked from content-centric networking. NDN router works on in path caching although other existing network architecture also having in-path caching mechanism but NDN works with the proper data structure and specialized interest packet handling mechanism which makes unique NDN architecture among several existing network architectures. It stores incoming interest packet address which in form of array of string containing alphanumeric values separated by ('/') delimited. While in Flat TCP/IP architecture IP address is used, which is not justifying today internet requirement where data handling is more important than machine and exactly NDN has hit this weakness of TCP/IP module, NDN follow only data address instead of machine address, but string is used for naming which leads to exhaustive memory consumption and extra lookup time for particular interest in table so proposed methodology reduces space complexity and lookup cost (i.e. time complexity). By using the encoding technique which potentially restrict the memory usage. In encoding technique encode the element of a name prefixes with an automatically generated unique number. This paper, based on reduction the lookup up cost and the insertion cost for a request.**

Keywords — Component Trie, Frequently recently used (FRU) algorithm, NDN

I. INTRODUCTION

The recently formed Named data networking (NDN) the upcoming internet architecture which is based on content-centric approach, and focuses on the content itself rather than "where the content is available". Two crucial efforts provided by the NDN: focus on "what data" required not where data is available and secure the data not the container. NDN packets transmit the names instead of source and destination address. Basically, NDN contains two types of the packet of communication, Interest packet and Data packet. NDN is a new architecture so it has many challenges like, but most promising challenges are size reduction of PIT and lookup and insertion cost in the PIT. In this paper discuss lookup and insertion cost in the PIT. NDN has many advantages over TCP/IP model. 1) NDN focus on data instead of IP address. 2) In TCP/IP model to get data, it needed to make the secure path and once you authenticate with the server, you trust the content, but in NDN we need to make the secure data instead of the path. So here data more secure in comparison to TCP/IP model. 3) NDN is stateful instead of stateless, in TCP/IP model request not store in router so it is stateless but in NDN, if it once forwarded the content it will Store that content till request will not satisfy. This paper, present how the lookup cost will reduce. Here no need to search Each and Every component in the component trie. Designed a time efficient scheme to reduce the lookup cost.

II. RELATED WORK

In existing NDN architecture need to search each and every request in the PIT from starting to end. Which is a time-consuming process to resolve the request. Many researchers gave a solution for lookup in NDN.

Saxena et.al [1], Radiant: they have proposed framework which consists of Name lookup module and Encoding module further Encoding module having two component name decomposition and Compact Radix Trie, each name prefixes decompose into individual component and assign token and these token now stored in PIT using Name Radix Trie by doing this they have claim memory efficiency but it will increase pre-processing complexity before storing named prefixes into PIT.

Yuan et.al [2] Reliably Scalable Name Prefix Lookup. In this paper proposed longest name prefix lookup design based on the binary search of hash tables organized by the numbers of name components in the prefixes. For forwarding rules that have up to k name components in each prefix, regardless of their specific characteristics, this design always guarantees at most log (k) hash lookups. But the pitfalls are in hash table collision problem will occur.

Yuan et.al [3] they have used the binary search of hash table which is used for IP address lookup in TCP/IP but they have used for NDN and proposed LNPM design which gives O(log(k)) in worst cases here k indicates a

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

number of components. They also extend their work with level pulling algorithm optimization of LNPM which works on observation but they issue with their module is arbitrarily they have chosen prefixes which is having length up to seven component which may fail in real time where average URL having length more than seven and also they are not static in length so their model may face problem while handling dynamic length of prefixes.

Miguel et al [4] IP Address Lookup Using a Dynamic Hash Function. They introduced IP address lookup algorithm that relates statistical analysis of the FIB to reduce memory usage for hash-based lookup. By introducing three dedicated data structures, index table, hash table, and small search group, here search space minimized by reducing the size of the lookup table. Using only one hash table not only limits the memory requirements but also reduces the complexity of the algorithm. To establish the lookup tables they proposed this algorithm.

Schneider et.al [5] IP Lookups Using Multiple ways and Multicolumn Search. To overcome from longest matching prefix problem they have used basic binary search technique which works on encoding a name prefixes at the starting and end of the range and again processes for best-matching prefix associated with a range. The less complexity of algorithm tends to be attractive for use for commercial purpose routers that have smaller routing tables. For IPv6, introduced a multi-column search technique that eliminates multiplicative factor of W/M inherent in basic binary search by selecting M bits columns wise for binary search, and jumping among columns using pre-computed information to obtain better measured.

III. PROPOSED WORK

This section, described our proposed framework for the PIT and will discuss the look-up time efficiency for PIT which reduces lookup cost i.e. Time complexity (amount of time, which is used for access to pending interest packet from PIT table). In this paper for matching the prefixes, TRIE data structure is used. Because TRIE data structure basically used for searching string prefixes efficiently and time complexity is O(M) where M is length of name prefixes while after using BST it will have (M*log N) time complexity here N indicates number of keys is used for BST, but here proposed work is to modified the Trie algorithm to make enhance lookup performance.

- Component trie
- Component Cache based on hash table technique
- Token Stack
- Frequency Array

1.Cache- when interest packet arrives, name prefixes decomposed into component. Now each and every component compared with existing prefixes in cache. If any component present, then it will hit and take the token with the help of corresponding address and increase the frequency in frequency map. If any component will miss then it will go to Component Trie and insert the component.

2.Compressed Component Trie- a kind of data structure which is basically used for name prefixes. It is compressed Trie.

3.Token frequency map is used to keep track of the frequency corresponding to token.

4.Stack is used for keeping the free token.

5.Encoded Name Trie is used for the encoded named which have to be retrieve from component Trie.

IV. PROPOSED MODEL

When interest packet arrives at NDN router and satisfies request on content store then NDN router response with the data packet. If request not found then go to PIT (with the framework) decompose the request in components and match the component in cache .if component matched in cache then fetch the token to the corresponding component with the help of address pointer. And go with frequency array and increase the frequency of token (which tokens are hit). If component is absent then go to component Trie and search the component. If component found then increase the frequency in frequency array by 1. If component not present, then insert that component and assign token (if token is available to stack, then pop the token and assign to new component, if not present then assign the new token incremented by 1.

V. AVERAGE ACCESS TIME FOR COMPONENT TRIE

$CT_{avg} = H (T_c) + (1-H) (T_c+T_t)$ + frequency increment cost.
CT_{avg}: Average time to access for component from Cache and trie.

ISSN (Online) 2394-2320
International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 12, December 2017

H: Probability to hit the component in cache
Tc: Time to access component in cache.
1-H: Probability to miss the component in cache.
Tt: Time to access component in component trie.

Component	Address Pointer
yahoo	500
gmail	200
in	400
com	100
www	300

Fig 1: Component Cache

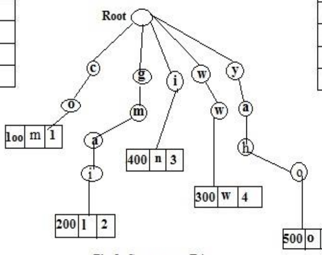


Fig 2: Component Trie

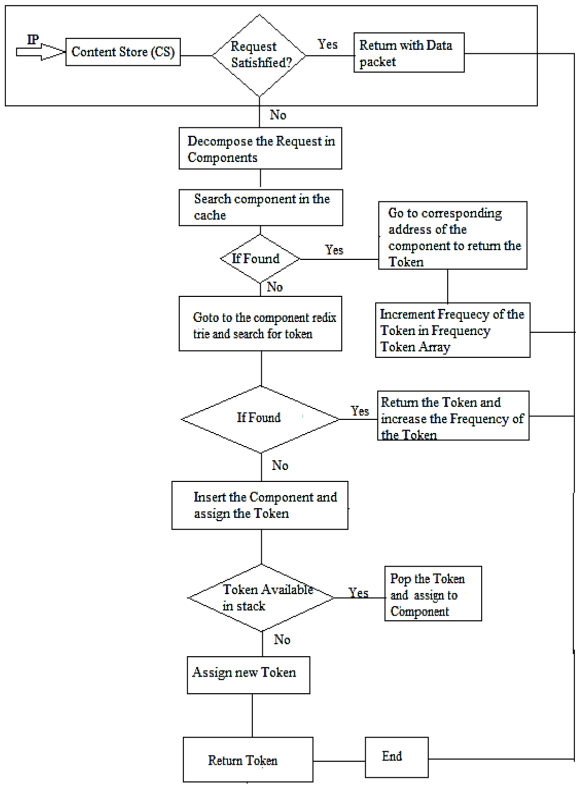
Token	Frequency
1	3
2	4
3	8
4	1
5	6

Fig 3: Frequency Array

Fig 4: Token stack

Frequency increment cost is time to update the frequency in the array. Which will be content so it will not add in lookup

VI. PREFIXES EVALUATION FLOW


IP
Content Store (CS)
Request Satishfied?
Yes
Return with Data packet
No
Decompose the Request in Components
Search component in the cache
If Found
Yes
Go to corresponding address of the component to return the Token
No
Goto to the component redix trie and search for token
Increment Frequecy of the Token in Frequency Token Array
If Found
Yes
Return the Token and increase the Frequency of the Token
No
Insert the Component and assign the Token
Token Available in stack
Yes
Pop the Token and assign to Component
No
Assign new Token
Return Token
End

Algorithm 1: Decomposed the request into components

```
input    : I: Interest message
             CS: Content Store
             PIT : Pending Interest Table
             FIB: Forwarding Interest Table
             data lifetime: Content lifetime
output   : DO: Data Object for each component DO:

    1.  if(component hit in cache==1)
    2.        frequency= frequency+1
    3.        return token
    4.    else
    5.        goto
    6.        component trie
    7.    if(component present in trie ==1)
    8.        frequency=frequency+1
    9.        return token
    10.  else
    11.    Insert the component and assign the new
           token to the component
```

VII. MATHEMATICAL ANALYSIS

For this analysis here, three parameters is taken which are as follow Page replacement algorithm, complexity, number of traversals.

A. Page hit ratio

When any request will arrive at pending interest table it will check on the component cache (part of the pit). In this condition, if the cache is full then apply the page replacement algorithm and calculate the hit ratio. Here three algorithms has taken into consideration which are described as First come first serve (FCFS), least recently used (LRU), frequency-based replacement (FBR). FBR is the proposed algorithm in our scheme. For simplicity, we assumed that cache size (CS) and no of components (NoC) respectively 5 and 20. Assume frequency of yahoo, gmail, in, com, www are 3, 4, 8, 1, 6 respectively. Assume 20 components are respectively yahoo, email, in, com, www, google, yahoo, email, in, com, www, google, com, www, in, google, yahoo, in, com, google.
Efficiency calculated for given data

$$\textbf{Efficiency } \eta = \frac{Hc}{Tc} \times 100$$

Hc = Number of hit component into Component Cache.
Tc = Total number of component.

B. Complexity

In our scheme, based on mainly to reduce lookup cost. To achieve this we proposed component cache based on Hash Table technique.
Time complexity for compressed Component Trie.

$$\sum_{i=1}^{N} = O(i*k)$$
$$= O\ (N*k)$$

C. Time complexity for component cache based on hash table technique

This section describe component cache based on hash table technique in our work. So if any component will search on component cache, it will give the result in constant time i.e. O(1).

CONCLUSION

Our work identified two kinds of problem in existing system 1) storing variable length consuming more memory. Which is not efficient for NDN router because NDN router has less memory to store a large amount of data so it will deplete soon so reduce this problem we will add new data structure in existing Scheme. 2) whenever required content requested or reply received by NDN router, an Exact matching technique is used for lookup to decide whether entry existing or not a particular instance of time. While the Existing system is checking Character by character which is time-consuming. To overcome this problem this paper proposed time efficient scheme for PIT lookup cost that is component replacement algorithm FRB.

REFERENCES

[1] Divya Saxena, Vaskar Raychoudhury, “Radient: Scalable memory efficient name lookup algorithm for named data networking”, Journal of Network and Computer Applications 63 (2016) 1–13.

[2] Haowei Yuan & Patrick Crowley, “Reliably Scalable Name Prefix Lookup”, Architectures for Networking and Communications Systems (2015)

[3] Huawei Yuan and Patrick Crowley, “Scalable Pending Interest Table Design: From Principles to Practice”, IEEE INFOCOM 2014 - IEEE Conference on Computer Communications. (2014)

[4] Miguel A .Martínez-Prieto, Nieves Brisaboa, Rodrigo Cánovas, Francisco Claude, Gonzalo Navarro, Practical “practical compressed string dictionaries”, Information Systems56(2016)73–108.

[5] klaus Schneider , Lixia Zhang, “A Practical Congestion Control Scheme for Named Data Networking”, ACM-ICN ’16, September 26–28, 2016, Kyoto, Japan 2016 ACM. DOI: http://dx.doi.org/10.1145/2984356.2984369

[6]Wei You, Bertrand Mathieu, Patrick Truong, Jean-Franc¸ois Peltier, Gwendal Simon, “Realistic Storage of Pending Requests in Content-Centric Network Routers” (2012).

[7] Xiaojun Nie David J. Wilson Jerome Cornet Gerard Damm Yiqiang Zhao, “IP Address Lookup Using A Dynamic Hash Function”, 0-7803-8886-0/05/2005 IEEE CCECE/CCGEI, Saskatoon, May 2005.

[8] Butler Lampson, Venkatachary Srinivasan, and George Varghese, “IP Lookups Using Multiway and Multicolumn Search”, IEEE/ACMTR ANSACTIONOSN N ETWORKINVGO, L.I , NO. 3, JUNE 1999.

[9] Stefanos Kaxiras Georgios Keramidas, “IPStash: a Power-Efficient Memory Architecture for IP-lookup” Proceedings of the 36th International Symposium on Microarchitecture (MICRO-36 2003).

[10] Matteo Virgilio, Guido Marchetto, Riccardo Sisto, “PIT Overload Analysis in Content Centric Networks”, ICN '13 Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking August 12 - 12, 2013