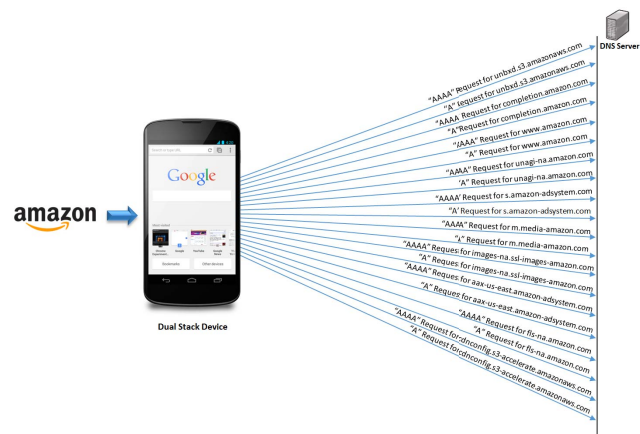


2nd Srihari Kuncha
Samsung R&D Institute India Bangalore
 Bangalore, India
 srihari.k@samsung.com

Index Terms—DNS; IPv4; IPv6; protocols; dual stack devices;

The Internet capable devices are increasing rapidly. This brings several types of resource problems with it. One of the major problems is limited number of IPv4 addresses. The IPv4 can only accommodate 4 billion unique IP address but it is predicted that by 2020 there would be 50 billion^[2] connected devices around the world. This problem was overcome by introduction of IPv6 which can now accommodate 3.4×10^{38} unique addresses. During early stages of the Internet era, every host had only an IPv4 address associated with it. But with the increasing demand for Internet capable devices, network providers are now capable of assigning both IPv4 and IPv6 addresses to device (dual stack device). Due to migration costs and technical difficulties, many hosts still remain to be IPv4 only capable. Since most of the devices have both IPv6 and

Fig. 1 shows how a dual stack device interacts with DNS server while loading amazon website from chrome browser.



Most of the devices are assigned with two DNS servers viz., primary and secondary. In an android dual stack device, whenever an application requests for host resolution, device first sends “AAAA” query to DNS server. Upon successful response device then sends “A” query to DNS server. In case of IOS, device sends both AAAA and A in parallel. Once the device gets response for both “AAAA” and “A” requests, it let’s the application know about the IP addresses it received.

Let us look at the DNS algorithm in android dual stack device w.r.t. both primary and secondary servers.

Algorithm 1 DNS Algorithm in Android dual stack device

```

1: Initialize  $i, j$  to 0
2: while  $i$  less than 2 do
3:   Send "AAAA" query to Primary DNS Server
4:   if response from PDS within 5 seconds then
5:     break
6:   end if
7:   Send "AAAA" query to Secondary DNS Server
8:   if response from SDS within 5 seconds then
9:     break
10:  end if
11:  Increment  $i$  by 1
12: end while
13: while  $j$  less than 2 do
14:   Send "A" query to Primary DNS Server
15:   if response from PDS within 5 seconds then
16:     break
17:   end if
18:   Send "A" query to Secondary DNS Server
19:   if response from SDS within 5 seconds then
20:     break
21:   end if
22:  Increment  $j$  by 1
23: end while
24: Send the response to application.

```

Whenever an application tries to open a website, it requests name resolution for multiple URL's. For each URL if device gets a response for "AAAA" then subsequent "AAAA" requests will not be sent. Similarly, if device gets a response for "A" request then subsequent "A" requests will not be sent. It takes 40 seconds to confirm that a DNS lookup has failed. The response might contain a valid IP address or a null IP address. The null IP address here signifies that the host is not capable of supporting the corresponding IP network. Once device gets response for both requests, IP addresses are forwarded to the requesting application. The application then opens a TCP session with the host.

Below Table provides the information on how DNS server responds to various queries it receives.

TABLE I: Requests and Responses from DNS server

Device IP Type	Server IP Type	DNS Query	DNS Response
IPv4	IPv4	A	Server IPv4
IPv4	IPv6	A	No Server
IPv4 & IPv6	IPv4 & IPv6	A	Server IPv4
IPv6	IPv4	AAAA	No Server
IPv6	IPv6	AAAA	Server IPv6
IPv6	IPv4 & IPv6	AAAA	Server IPv6
IPv4 & IPv6	IPv4	A & AAAA	Server IPv4
IPv4 & IPv6	IPv6	A & AAAA	Server IPv6
IPv4 & IPv6	IPv4 & IPv6	A & AAAA	Server IPv4/IPv6

B. Motivation and importance of Q-DNS Solution

A dual stack device sends "AAAA" and "A" requests sequentially for DNS lookup irrespective of host supporting the network. In case host is IPv4 only capable, DNS server responds only to "A" queries. Due to initial "AAAA" request device sends, DNS resolution time is high in comparison with devices having IPv4 only address. This also causes additional DNS traffic for every "AAAA" request we send. Similar situation can also be observed when host supports IPv6 only.

Below Fig. 2 explains, how a DNS server responds when a dual stack device tries to load a IPv4 only host and IPv6 only host respectively.

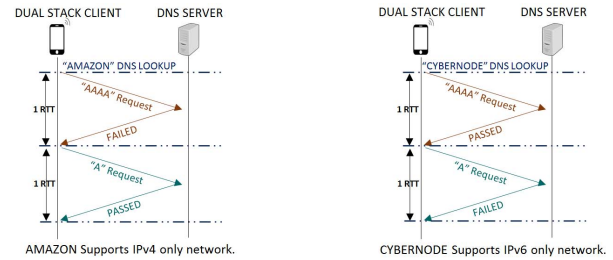


Fig. 2. Dual Stack device behavior w.r.t different hosts

We conducted several experiments to understand on how DNS Latency differs from a dual stack device with IPv4 only device. We have collected data for some of the most popular Indian websites like Myntra, Flipkart, Amazon, Paytm etc. Experiments were conducted with R-JIO network. We have categorized our experiments into two groups. One group is a set of five frequently visited hosts and other group is a set of four infrequently visited hosts.

Fig. 3 shows, how a IPv4 only device interacts with DNS Server while loading amazon website from chrome browser.

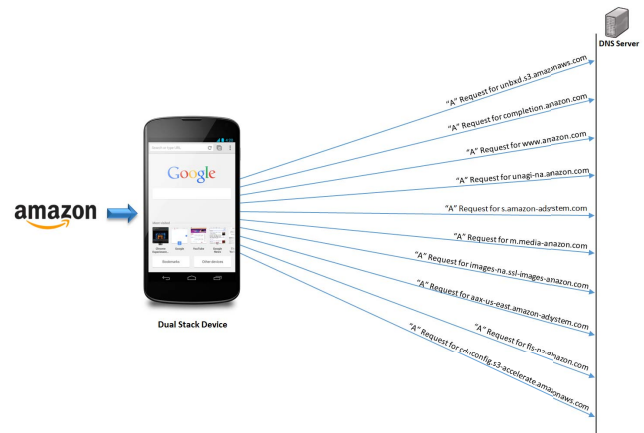


Fig. 3. Total queries launched while loading Amazon with Q-DNS

Fig. 4 and Fig. 5 depict average DNS Latency for frequently visited and infrequently visited hosts. Through above experimentation we have concluded that given hosts are IPv4 only

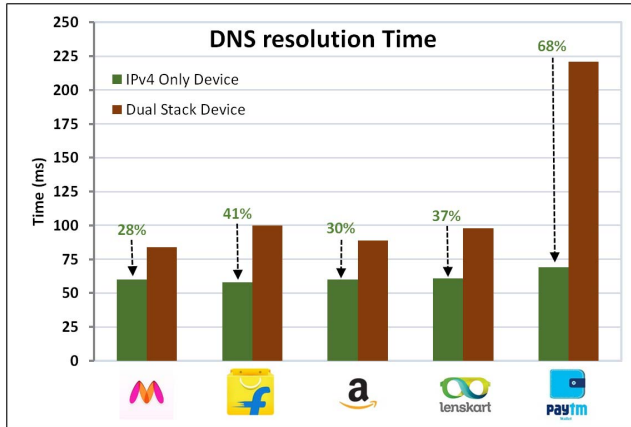


Fig. 4. DNS resolution time for frequent websites

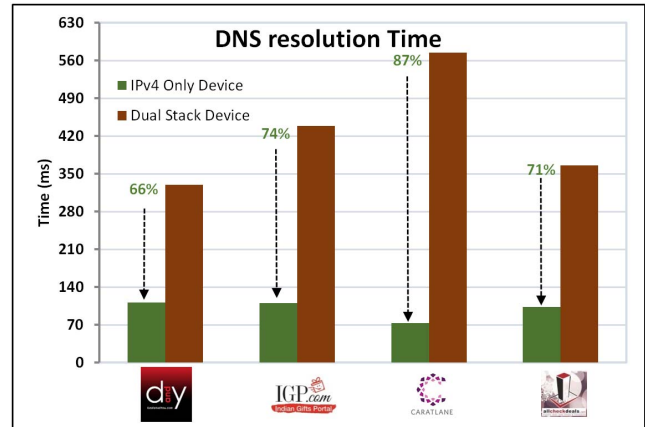


Fig. 5. DNS resolution time for infrequent websites

capable. So, in dual stack device sending a “AAAA” causes extra DNS traffic, and consumes more time before starting a TCP session which leads to a bad user experience. By looking at the graphs we can conclude that DNS Latency is less in IPv4 only device when compared to dual stack device.

To overcome the above explained problem and improve user experience we implemented Q-DNS in Android Platform. Q-DNS was able to reduce DNS Latency by 30% to 60% and DNS traffic by 50% in dual stack devices.

C. Novelty and Practical Importance

- Q-DNS is a light weight device only solution which reduces DNS Latency and load on DNS server.
- It does not require any changes in existing software, hardware or standard network protocols.
- It is also a platform independent solution and can be deployed across any platform like Android, IOS, Linux.
- It can be implemented in many devices like Smart Phones, Desktop, Tablets.

Improving application responsiveness by reducing latency is one of the most researched topics for Next generation networks in recent times. Reducing Latency by a few hundred milliseconds can increase a company revenue by millions of dollars. For Google, a delay of 400 ms in search responses would have lost them \$75M^[11]; and for Bing, 500 ms of latency decreases revenue per user by 1.2%. Hence, our attempt to reduce DNS latency which might vary from 200 ms to 5 sec based on network conditions is a positive move and hence it is a value added solution to improve user experience.

II. RELATED WORK

To solve DNS latency issue and provide fast connectivity in dual stack devices, IETF had proposed an algorithm known as “Happy Eyeballs”(also called Fast fallback). The algorithm and its requirements are described in RFC 8305. Happy Eyeballs algorithm has one primary goal. Providing fast connection for users, by quickly attempting to connect using IPv6 and if that connection attempt is not quickly

successful, fallback and try to connect using IPv4. Happy Eyeballs implementation can be seen in MAC OS, where both “AAAA” and “A” requests in parallel. An application that uses a Happy Eyeballs algorithm checks for both IPv4 and IPv6 connectivity and uses the first connection that is returned by the network. Preference is given to IPv6 networks over IPv4 networks.

Even though the above solution tries to reduce Latency it sends both “A” and “AAAA” request which might not be always necessary thus increasing load on the network.

Q-DNS is completely different and a unique solution to solve DNS latency issue in dual stack devices with zero overhead. It is a client only solution that learns the host capabilities and drops unnecessary requests at the device end.

III. PROPOSED Q-DNS SOLUTION

Q-DNS provides an intelligence to device by dropping “AAAA” requests when host is IPv4 only capable and similarly dropping “A” requests when host is IPv6 only capable. It solves the problem by learning the host capabilities at device side.

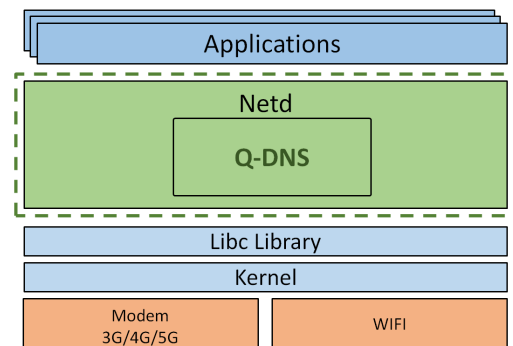


Fig. 6. Software Architecture of Q-DNS

A. Why is Q-DNS solution better than DNS Cache

DNS cache is a dynamic database maintained by device operating system, which is used by applications for a quick

look up. The most important attribute for a DNS record to survive in DNS Cache is Time to Live. Typically this attribute is set by network and ranges from 2(sec)-1(day)^[7]. Suppose we opened www.myntra.com for first time. Device sends both “AAAA” and “A” request to DNS server. Server responds with IPv4 only address as IPv6 is not supported by the host. Device caches IP address, address type, TTL etc. Suppose user wants to open myntra again after some time. By this time record might have already been deleted and device requests DNS server again with both “AAAA” and “A” request. This can be improved if device is intelligent enough to capture that host is IPv4 only capable. So, whenever we encounter a new host Q-DNS learns host capability and kind of network it supports. This helps not to generate unnecessary requests, thus adding extra benefit over Caching.

B. Q-DNS Algorithm Approach

Q-DNS algorithm resides on device and generates those requests for which it has learned that response carries a proper IP address. For IPv4 only hosts, “A” query will be sent and for IPv6 only hosts, “AAAA” will be sent. Both “A” and “AAAA” queries will be sent for hosts supporting IPv4 and IPv6. A hash Map is used for classification purpose. When Q-DNS algorithm encounters a new host, it sends both requests to DNS server. Upon successful response from DNS, host is categorized based on the IP addresses it received. The learning will be carried out for few iterations before we finally categorize the host. This is required as DNS server might not respond with actual capabilities of the host in first go.

Fig.7 and Fig.8 shows the snippets of TCP dump analysis on Wireshark tool while loading myntra website from chrome browser.

61	5.611819	10.102.146.207	49.45.0.1	DNS	76 Standard query 0xec7f AAAA www.myntra.com
62	5.655930	49.45.0.1	10.102.146.207	DNS	215 Standard query response 0xec7f AAAA www.myntra.com CNAME www.myntra.com
63	5.656238	10.102.146.207	49.45.0.1	DNS	76 Standard query 0xd084 A www.myntra.com
64	5.686864	49.45.0.1	10.102.146.207	DNS	173 Standard query response 0xd084 A www.myntra.com CNAME www.myntra.com

Fig. 7. Device without Q-DNS sending both requests

124	5.357135	10.102.146.207	49.45.0.1	DNS	76 Standard query 0x4179 A www.myntra.com
125	5.624269	49.45.0.1	10.102.146.207	DNS	173 Standard query response 0x4179 A www.myntra.com CNAME www.myntra.com

Fig. 8. Device with Q-DNS sending only “A” requests

C. What if Host capabilities change in future

Suppose, we have categorized “www.amazon.com” to be IPv4 only based on Q-DNS algorithm and in the near future amazon wants to shift from IPv4 only to IPv6 only or support both IPv4 and IPv6 networks. How does Q-DNS get to know the changes in host capabilities?

The above problem is taken care by introducing a timer w.r.t to every host. Whenever we encounter a new host we assign a timestamp to it. While DNS Lookup, if the timestamp crosses a threshold we send both “A” and “AAAA” requests for host resolution. Based on response we update the hash map.

D. Memory constraints of using HashMap

To be a client only solution device needs to add an extra amount of memory to store the data. There is always a trade of between time complexity and space complexity. With increase in demand for Time optimization solutions we used enhanced version of HashMap known as LinkedHashMap which is a memory-hungry collection in JDK.

LinkedHashMap extends HashMap by using LinkedHashMap.Entry as an entry in internal array of entries. LinkedHashMap.Entry extends HashMap.Entry by adding “prev” and “next” pointers, thus implementing a linked deque. LinkedHashMap.Entry consumes 40 bytes of data. We therefore used dynamic allocation of data whenever there is requirement for new entry instead of allocating a static Map. Let S be the number of entries in LinkedHashMap then the total memory M allocated is given by

$$M = S * 40bytes \quad (1)$$

Algorithm 2 Q-DNS Algorithm

Input: host h , Servers d , CurrentTime t , ThresholdTime T
Output: host IP addresses

```

1: procedure GETADDRINFO
2:    $type \leftarrow PF\_V4/V6$ 
3:    $t \leftarrow System.time$ 
4:   if  $h$  is present in Hash Map then
5:      $prevtype \leftarrow h.type$ 
6:     if  $(t - h.time) \leq T$  then
7:       if  $h$  supports V4 Only then
8:          $type \leftarrow PF\_V4$ 
9:       else if  $h$  supports V6 Only then
10:         $type \leftarrow PF\_V6$ 
11:      else
12:         $type \leftarrow PF\_V4/V6$ 
13:      end if
14:    else
15:       $h.time \leftarrow t$ 
16:    end if
17:  end if
18:  if  $type$  is  $PF\_V4$ : “A” requests
19:  else if  $type$  is  $PF\_V6$ : “AAAA” requests
20:  else: Both requests
21:  end procedure
22:  Send requests to  $d$ 
23:  Receive IP addresses for  $h$ 
24:  Learn about  $h$  based on IP addresses received
25:  if  $h$  is present in Hash Map then
26:     $newtype \leftarrow h.type$ 
27:    if  $newtype \neq prevtype$  then
28:      Update result for  $h$  in hashmap
29:    end if
30:  else
31:    Add  $h$  into Hash Map
32:  end if

```


E. Performance Analysis Model

We have analyzed the expected improvement of DNS Latency by using the following Model.

Let T_b be the time taken for DNS resolution without Q-DNS

$$T_b = R_a + R_{aaaa} \quad (2)$$

Where, R_a is RTT for “A” request and R_{aaaa} is RTT for “AAAA” request. Let T_a be the time taken for DNS resolution with Q-DNS

$$T_a = \rho_a R_a + \rho_{aaaa} R_{aaaa} + T_h \quad (3)$$

Where ρ_a and ρ_{aaaa} are the probabilities of the host being IPv4 only and IPv6 only respectively. Both are Bernoulli distribution which can have value of either 0 or 1. T_h is the time taken to search for an entry in LinkedHashMap.

For a HashMap to be optimistic it needs to have a better Hash function which can minimize the number of collisions. We have used Polynomial accumulation Hash function which combines the character values (ASCII) $a_0 a_1 \dots a_{n-1}$ by viewing them as the coefficients of a polynomial. The polynomial is computed with Horner’s rule, ignoring overflows, at a fixed value x :

$$a_0 + x(a_1 + x(a_2 + \dots x(a_{n-1} + x * a_{n-1}) \dots)) \quad (4)$$

The choice $x = 33, 37, 39$, or 41 gives at most 6 collisions on a vocabulary of 50,000 English words^[12].

Let us try to compute the value of T_h w.r.t Intel Core i7 7500U processor which does 49,360 million instructions per second (MIPS) at 2.7GHz^[13] to show that the value doesn’t have any significance in computing total DNS resolution time with Q-DNS. Computing T_h is basically a combination of two operations. One is computing hashvalue(τ_h) and the other is searching(τ_s) for the entry at that hashvalue from LinkedHashMap.

$$T_h = \tau_h + \tau_s \quad (5)$$

Let us assume we have to get the value of a N bit key from LinkedHashMap then

$$\tau_h = N * \Theta(n^2) \quad (6a)$$

$$\tau_s = N * 6(MaximumCollisions) \quad (6b)$$

Where, n is 32 bit integer and $\Theta(n^2)$ is computational complexity of multiplication. This values are with respect to the above mentioned Hash function. If we assume the value of N to be 1000 bits. Then the value of T_h comes out to be 1.1 million instructions which can be computed in microseconds. Hence, we shall ignore the time taken to search an entry from LinkedHashMap.

The total improvement(τ) of DNS latency by Q-DNS solution can be given by

$$\tau = (T_b - T_a) \quad (7)$$

After simplification,

$$\tau = (1 - \rho_a)R_a + (1 - \rho_{aaaa})R_{aaaa} \quad (8)$$

IV. PERFORMANCE EVALUATION OF Q-DNS

The proposed solution is successfully implemented in Samsung Galaxy S7 device. The experiment was performed on two dual stack devices, one with Q-DNS algorithm and other without Q-DNS algorithm. Both devices were configured with same hardware, software and experiment was performed on R-JIO cellular network.

We have categorized our study into 2 groups. First group is a set of hosts, which support IPv6 only networks. Second group is a set of hosts, which support IPv4 only networks. We further categorized second group into 3 subgroups: Popular American Hosts; Popular Indian Hosts; Popular Korean Hosts. Fig. 9 shows percentage gain on DNS latency of IPv6 only hosts.

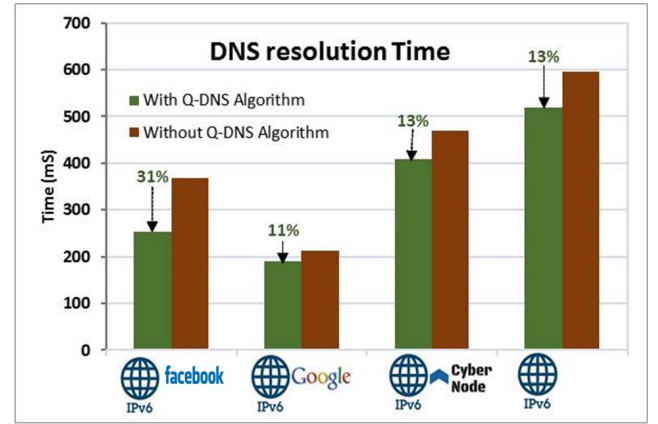


Fig. 9. Performance Evaluation for IPv6 only hosts

DNS lookup varies based on network conditions. Hence, for the second group, we have taken results in low speed and high speed Internet conditions. Fig. 10, Fig. 11 and Fig. 12 depict the percentage gain on DNS latency of Indian hosts, American hosts and Korean hosts respectively.

Application	High Speed Internet			Low Speed Internet		
	With QDNS (ms)	WithOut QDNS (ms)	Percentage Gain	With QDNS (ms)	WithOut QDNS (ms)	Percentage Gain
amazon primevideo	945	1487	61%	35894	43933	18%
lenskart.com	185	220	15%	16295	26518	38%
flipkart.com	160	740	78%	15859	19829	20%
paytm	405	658	61%	31437	43348	27%
cricbuzz	91	250	63%	9760	16374	40%
Quikr	2108	3097	31%	2606	3019	13%
yatra	208	307	32%	519	706	26%
rediff	2824	4961	43%	23317	27626	15%
hotstar	1958	3051	35%	8113	10075	19%

Fig. 10. Performance Evaluation for Popular Indian Hosts

In case of Indian websites, DNS Latency can be improved by 30%-60% in High Speed Internet conditions and 20%-40% in Low Speed Internet conditions.





Application	High Speed Internet			Low Speed Internet		
	With QDNS (ms)	Without QDNS (ms)	Percentage Gain	With QDNS (ms)	Without QDNS (ms)	Percentage Gain
	377	567	33%	18650	32658	43%
	1814	2897	37%	45374	92887	51%
	875	1051	17%	13365	39343	66%
	4187	5847	28%	40890	96605	57%

Fig. 11. Performance Evaluation for Popular American Hosts

In case of American websites, DNS Latency can be improved by 20%-40% in High Speed Internet conditions and 40%-65% in Low Speed Internet conditions.





Application	High Speed Internet			Low Speed Internet		
	With QDNS (ms)	Without QDNS (ms)	Percentage Gain	With QDNS (ms)	Without QDNS (ms)	Percentage Gain
	704	1137	38%	8136	12197	33%
	6056	7368	18%	28394	54984	48%
	22260	30751	28%	22439	37757	41%
	3582	4365	18%	22970	30082	24%

Fig. 12. Performance Evaluation for Popular Korean Hosts

In case of Korean websites, DNS Latency can be improved by 20%-40% in High Speed Internet conditions and 25%-50% in Low Speed Internet conditions. In all of the above scenarios DNS requests were reduced by 50%.

Our new approach helped in dropping “AAAA” requests in case of hosts being IPv4 only and dropping “A” requests in case of hosts being IPv6 only. This resulted in reducing DNS resolution time and DNS traffic

V. CONCLUSION

ISP’s normally assign both IPv6 address and IPv4 address to device. Device having both addresses, tries to send “AAAA” and “A” request to DNS server, which might not be always necessary. This paper proposes an algorithm which learns the host capabilities at device side and sends DNS request for which it is sure of getting a valid response. Our Solution is a light weight client only solution. It can be easily deployable across platforms like Android; Linux; IOS, without any change in existing software, hardware or network protocols. The proposed solution was deployed on Samsung Galaxy S7 dual stack device. It was later evaluated on various hosts under low speed and high speed Internet conditions. The results indicate that device with Q-DNS out performs the existing logic and reduces the DNS Latency by 30% to 60% as well as traffic on DNS server by 50%.

For platforms like IOS, where both “AAAA” and “A” requests are sent in parallel, the proposed solution reduces traffic on DNS server. For platforms like Android and Tizen where the requests are sent sequentially there is an additional benefit of reduced DNS Latency.

REFERENCES

- [1] Jaeyeon Jung, Hari Balakrishnan: DNS Performance and the Effectiveness of Caching in nms.lcs.mit.edu/papers/dns-ton2002.pdf
- [2] <http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated>
- [3] M. Mawatari, M. Kawashima, C.Byrne: 464XLAT: Combination of Stateful and Stateless Translation
- [4] <https://umbrella.cisco.com/blog/2009/12/16/20-billion-queries/>
- [5] Jun Bi, Jianping Wu IPv4/IPv6 Transition Technologies and Univer6 Architecture in <http://paper.ijcns.org>
- [6] V.M. Paul and J.D. Kevin: Development of Domain Name System in cseweb.ucsd.edu/classes/wi01 pg: 123-126
- [7] Chris Gonyea: Everything you ever wanted to know about TTLs: DNS traffic management: <https://dyn.com/blog/dyn-tech-everything-you-ever-wanted-to-know-about-ttls/>
- [8] Robert Beverly, Nicholas Weaver, Arthur Berger Inferring Internet Server IPv4 and IPv6 Address Relationships from MIT labs edu.
- [9] Google: Studing about the importance of Primary and Secondary DNS Servers in webhostinggeeks.com/blog/understanding-the-difference-between-a-primary-and-secondary-dns-server/
- [10] Hamza Boulakhirf: Analysis of DNS Resolver Performance Measurements
- [11] B. Briscoe, Reducing internet latency: A survey of techniques and their merits, IEEE Communications Surveys and Tutorials, vol. 18, no. 3, pp. 2149-2196, 2014.
- [12] Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015
- [13] Wikipedia, the free encyclopedia, InstructionsPerSecond