IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Reducing User Perceived Latency in Smart Phones Exploiting IP Network Diversity

JAMSHEED MANJA PPALLAN[1], SWETA JAISWAL[1], KARTHIKEYAN ARUNACHALAM[1], PASQUALE IMPUTATO[2], STEFANO AVALLONE[2], DRONAMRAJU SIVA SABAREESH[1], and MADHAN RAJ KANAGARATHINAM[1],

[1]Mobile Communication R&D, Samsung R&D Institute India - Bangalore (e-mail: {jamsheed.mp, sweta.j, karthikeya.a, s.sabareesh, madhan.raj}@samsung.com)
[2]Computer Engineering Department, University of Naples Federico II, Italy (e-mail: {pasquale.imputato, stefano.avallone}@unina.it)

Corresponding author: Jamsheed Manja Ppallan (e-mail: jamsheed.mp@samsung.com).

ABSTRACT The Fifth Generation (5G) wireless networks set its standard to provide very high data rates, Ultra-Reliable Low Latency Communications (URLLC), and significantly improved Quality of Service (QoS). 5G networks and beyond will power up billions of connected devices as it expands wireless services to edge computing and the Internet of Things (IoT). The Internet protocol suite continues its evolution from IPv4 addresses to IPv6 addresses by increasing the adoption rate and prioritizing IPv6. Hence, Internet Service Providers (ISP's) are using the address transition method called dual-stack to prioritize the IPv6 while supporting the existing IPv4. But this causes more connectivity overhead in dual-stack as compared to the single-stack network due to its preference schema towards the IPv6. The dual-stack network increases the Domain Name System (DNS) resolution and Transmission Control Protocol (TCP) connection time that results in higher page loading time, thereby significantly impacting the user experience. Hence, we propose a novel connectivity mechanism, called NexGen Connectivity Optimizer (NexGenCO), which redesigns the DNS resolution and TCP connection phases to reduce the user-perceived latency in the dual-stack network for mobile devices. Our solution utilizes the IP network diversity to improve connectivity through concurrency and intelligent caching. NexGenCO is successfully implemented in Samsung flagship devices with Android Pie and further evaluated using both simulated and live-air networks. It significantly reduces connectivity overhead and improves page loading time up to 18%.

INDEX TERMS Domain Name System, TCP/IP, Wireless Communication, Mobile Networks, Connectivity, Page Loading Time, Android Platform,

I. INTRODUCTION

THE Next Generation Networks (NGN) expand wireless services beyond mobile internet to critical communications segments such as edge computing and the Internet of Things (IoT). With the advent of 5G networks and beyond, the number of connected devices will upsurge in the coming years. The emerging 5G technology empowers these connected devices, including smartphones, with a wide range of application scenarios from low latency communications to very high data rate communications. As low network latency is one of the Key Performance Indicator (KPI) for 5G networks, it is relevant to reduce the network latency overhead of Internet protocol suite as much as possible.

Internet protocol suite has started to evolve to meet the future needs of NGN. The Internet Engineering Task Force (IETF) began in 1999 to define the need for larger addressing space and, the effort went in the definition of the newer IPv6. The IPv6 provides a vast addressing space with added advantages such as support for device security, mobility, and configuration. According to [1], the number of connected IoT devices will reach around 42 billion units by 2022, and hence, relying on IPv6 addresses is inevitable. Also, Google's data [2] presents a fast IPv6 network adoption rate, which is doubling every six months and following an exponential curve, as shown in Fig. 1.

Mobile Network Operators (MNO's) are expanding their networks using IPv6 address space. At the same time, they are assuring the existing IPv4 compatibility using the address

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

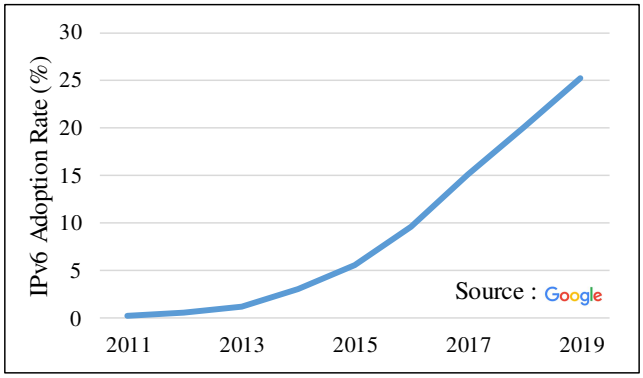

FIGURE 1. IPv6 adoption rate since 2011. The IPv6 adoption rate is doubling every six months and following an exponential curve.

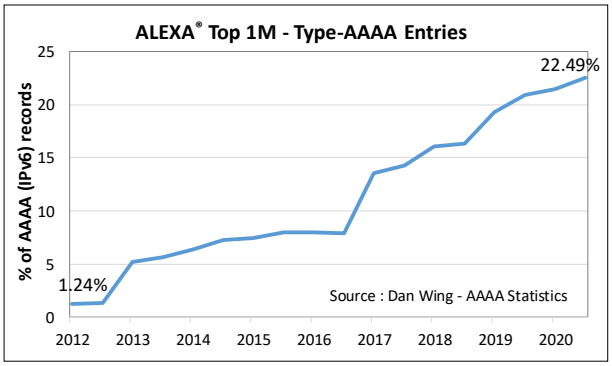

FIGURE 2. Top 1 million websites by Amazon's ALEXA ranking with AAAA DNS records. The fraction of websites among the top 1 million websites by Amazon's ALEXA ranking that have AAAA entries in the DNS servers reached about 22.5% as of May 2020.

transition method called *dual-stack* network architecture. The need to support IPv4 network arise since a relevant number of resources are reachable only through the IPv4 network. Indeed, the World IPv6 Launch Day in 2012 [3] encouraged several notable content providers to start providing services over both IPv6 and IPv4. Currently, the fraction of websites among the top 1 million websites by Amazon's ALEXA ranking [4] that have AAAA (IPv6) entries in the DNS servers is about 22.5% as of May 2020 [5] (refer to Fig. 2). Also, even though IPv6 addresses adoption is exponentially increasing, the fraction of web page elements over IPv6 that fails to load is still significant. Studies show that 27% of websites with AAAA entries have web page elements that fail to load in IPv6 [6]. Hence, any algorithm that prioritizes the path selection in favor of IPv6 may end up in connecting to the slower path, which needs to be revised.

The dual-stack network architecture leads to support both IP versions (IPv4/IPv6) and leaves the implementation to Operating System (OS) for a possible preference schema towards an IP version. For instance, Android and iOS-based mobile devices support IPv4/IPv6 domain name resolution and server connection with a preference schema towards IPv6. In Android devices, the dual-stack architecture is supported through a priority resolution of a domain name for an IPv6 host and then for an IPv4 host. Then Android lets the application to create a connection with the resolved host. However, for dual-stack network-supported Android devices, the DNS lookup and TCP connection establishment take more time compared to single stack (IPv4-only) devices. For instance, in case of DNS fails over IPv6, the application waits a long time before it falls back to IPv4. Apple tried to solve this problem in iOS devices by implementing the IETF proposed algorithm known as Happy Eyeballs (HE) [7]. The HE algorithm triggers both Type-AAAA (IPv6) and Type-A (IPv4) DNS queries in parallel and establishes TCP connections simultaneously. HE introduces delay timers (to wait) in favor of IPv6 when an IPv4 resolution/connection is available before an IPv6 resolution/connection. The idea is to introduce by design a preference schema towards IPv6 to support the IPv6 adoption. However, currently, it is not clear if a preference schema towards IPv6 can keep the user-perceived latency under control (or even improve it).

We conducted an experimental campaign to highlight possible unfavorable points towards a pure IPv6 preference schema in the connection phase. We tested the top 300 websites, with A and AAAA entries in the DNS server (among the 21% of websites which have both entries), from Alexa ranking over 100 trials per day for 10 days. The idea was to resolve the domain name with IPv4 /IPv6 addresses and then test the performance in terms of the initial RTT of both TCP servers. We introduced a term called TCP slowness, defined as the difference in TCP connection time of IPv4 and IPv6 servers for a (pre) resolved domain, $\Delta s_{tcp}(u) = t_{v4}(u) - t_{v6}(u)$, where u is the domain to be loaded. Positive values of Δs_{tcp} denote that IPv6 connections are faster and negative values of Δs_{tcp} denote that IPv4 connections are faster. Then, we defined $\Delta s_{tcp} = 0$ in case of an IPv6 connection failure. The results in Fig. 3 clearly show that about 60% of IPv4 connections are faster than IPv6 connections. Another important observation is that around 7% of IPv6 connections were failing, which is shown as $\Delta s_{tcp} = 0$ in the TCP slowness graph. Hence, the results prove that i) IPv6 connections are often slower than IPv4 and that ii) the number of IPv6 connection failures is still not negligible. Based on the current connectivity overhead, giving a higher priority to IPv6 in the connection phase will bring a higher user-perceived delay, which substantiates our claims about preference schema towards IPv6.

We further analyzed Android dual-stack devices and found additional bottlenecks, especially in network or link failure cases. For example, in case of a broken link or blocked address family (IPv4 or IPv6), multiple TCP connections attempts in sequence cause higher user-perceived delay in dual-stack devices. Moreover, since DNS cache is not effective on Android due mainly to communication overhead among apps and cache, sequential DNS lookup (AAAA followed by A) and shorter TTL (Time to Live) values of DNS resource records

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

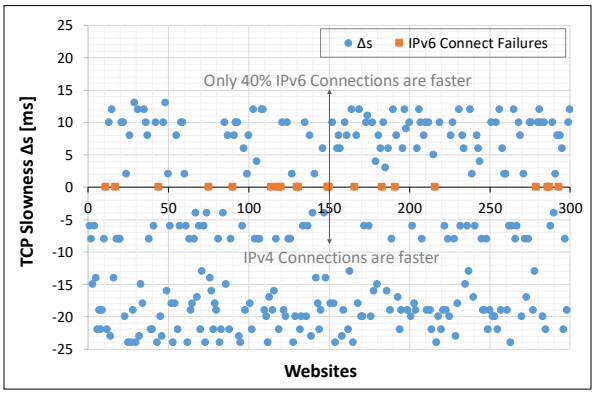

FIGURE 3. TCP Slowness (Δs_{tcp}) analysis in TCP connection time. Positive values of Δs_{tcp} denote that IPv6 connections are faster and negative values of Δs_{tcp} denote that IPv4 connections are faster. The analysis shows about 60% of TCP connections over IPv4 are faster than TCP connections over IPv6.

degrade client application's performance during slower network conditions [8].

In this work, we propose a novel solution called *NexGen Connectivity Optimizer (NexGenCO)* for improving the connectivity of the dual-stack Android mobile devices. Our proposal aims to support the co-existence of IPv4 and IPv6, keeping into account the user-perceived latency. At this end, NexGenCO relies on concurrency between IPv4 and IPv6 connectivity and intelligent caching of resolved domain names.

Following are the main benefits of NexGenCO:

- is the first-ever implementation of asynchronous DNS resolution and application-specific DNS cache on the Android platform;
- overcomes the drawbacks of DNS resolution and TCP connection on dual-stack networks;
- proposes a simple and efficient methodology for estimating the best TCP connection at any moment and maps it to the application;
- significantly reduces connectivity overhead and improves application page loading time up to 18% consistently.

The rest of the paper is organized as follows. Section 2 describes background and related work. In Section 3 we describe our proposal. Then, in Section 4, we introduce a model for performance evaluation. In Section 5 we present the performance evaluation in controlled environment. Then, in Section 6 we present the performance evaluation in apps. Finally, in Section 7 we conclude the work.

II. BACKGROUND AND RELATED WORK

A dual-stack host with IPv6 connectivity always prefers an IPv6 path. During domain name resolution, DNS responses get filled with a list of hosts in an order that prefers IPv6. Later in the connection phase, the application connects to an IPv6 host concerning the DNS response order. However, the preference for an IPv6 path can increase the user-perceived latency due to the higher DNS resolution and TCP connection delay over IPv6. Also, the application responsiveness reduces further when there are connection failures over IPv6.

Android implements the base dual-stack architecture in its devices for supporting the dual-stack networks (traditional dual-stack in Fig. 4). The system function `getaddrinfo` resolves the domain name into a list of hosts with a preference towards IPv6. Then, the system lets the application to connect to a host through the function `connect`. Android system stack does not have a dedicated process for maintaining a proper DNS cache [9]. It maintains a DNS cache based on the smallest TTL value of the DNS response. Every process running on the Android system communicates with DNS cache using Inter-Process Communication (IPC), which creates additional overhead in the mobile device. It is worth noting that most of the popular Content Distribution Network (CDN) providers set a minimum TTL value in DNS resource records. This helps CDN providers to handle system fail-over and to distribute load among the servers. These shorter TTL valued DNS queries expire early, which creates additional DNS queries in the network and also increases the load on the DNS servers [10]. Hence, avoiding unnecessary DNS resolution using an enhanced cache would be very efficient.

In iOS, Apple implements an algorithm named Happy Eyeballs (version 1 [11] and version 2 [7]) for improving the connectivity of dual-stack devices (shown in Fig. 4 as Happy Eyeballs). The algorithm introduces the idea of *concurrency* between IPv4 and IPv6. The HE algorithm, by design, prefers its path selection in favor of IPv6. It introduces a static wait timer for Type-`AAAA` DNS response before processing Type-`A` DNS response. Similarly, it has a static wait timer for IPv6 connection before falling back to IPv4 connection attempt. These delays are referred to as the *resolution delay* (`rd`) and *connection attempt delay* (`cad`) respectively. HE caches information regarding the outcome of each connection attempt, and it uses that information to avoid thrashing the network with subsequent attempts. These cache entries should be flushed when their age exceeds a system-defined maximum on the order of 10 minutes. As mentioned in [12], even after reducing the HE timer to 150 ms, it maintains the same IPv6 preference levels. HE prefers IPv6 connections 90% of the time even if it is slower than IPv4. Further, the study states that only 40% of top 10,000 websites by ALEXA ranking are faster over IPv6 [12].

A few works available in the literature focus on the idea of *pre-fetch* a resolution for a domain name or a *pre-connect* towards a server. However, all the proposals do not explore the type of IP network (IPv6/IPv4) for reducing the overall user-perceived delay. Chromium and Mozilla are following a technique [13] to pre-fetch DNS queries and reduce user-perceived latency. But this idea pre-fetches the DNS queries based on browsing history. In [14], Qualcomm proposes the pre-resolution of a domain name based on browsing history. Google published a patent [15] for reducing network connection latency and navigation latency, which also requires user interaction as input and is limited to browser applications. In previous works, we explored optimization mechanisms in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

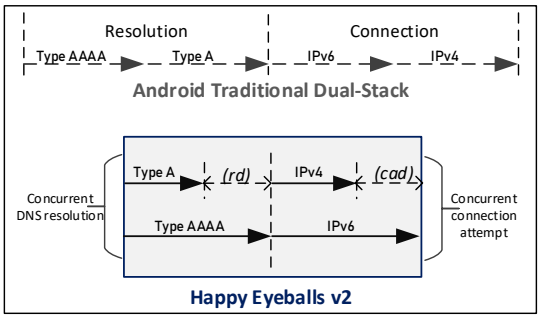

FIGURE 4. Traditional dual-stack implemented in Android and Happy Eyeballs implemented in iOS. Android waits for resolution failure or connection failure before to fall on IPv4 while iOS waits till static resolution delay or connection attempt delay are expired before to fall on IPv4.

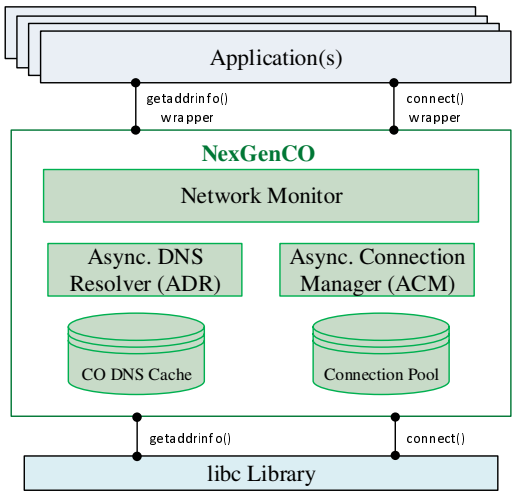

FIGURE 5. Software architecture of NexGenCO. It is a shim layer between the libc library and the application layer for intercepting DNS lookup and TCP connect for improving the connectivity.

Android to reduce the latency overhead. In [16] we introduced in Android the idea of pre-resolution of a domain name and intelligently cached the results to reduce the resolution delay. Then, in [9], we introduced the idea of pre-connect to a host to reduce the connection delay. On the iOS side, HE does not provide any specification of pre-resolution and pre-connect mechanisms.

In this solution, we considered all the limitations of previous works and proposed NexGenCO for reducing the connectivity latency overhead and improving the performance of the applications in terms of page loading time. NexGenCO proposes first of its kind connectivity mechanism that is completely different from these earlier works. Indeed, it does not rely on any user inputs and and is not specific to any application or protocol. It is a client-only software layer and provides a platform-independent solution.

III. NEXT GENERATION CONNECTION OPTIMIZER

In the following, we provide a detailed description of the proposed solution. NexGenCO aims to reduce page loading time on mobile devices by exploiting IP network diversity. The solution relies on concurrent operations over IPv4 and IPv6. It performs DNS resolution and TCP connection concurrently over IPv4 and IPv6. NexGenCO avoids the application overheads by implementing ahead-of-time DNS resolution and TCP connection. Also, it introduces per-app caching mechanism for storing DNS responses and TCP connection descriptors.

A. SOFTWARE ARCHITECTURE

NexGenCO is a client software shim layer which lies between the `libc` library and the application layer (Fig. 5). The layer provides wrapper functions for `getaddrinfo` and `connect` libc functions. NexGenCO comprises of three modules, namely, *Asynchronous DNS Resolver* (ADR), *Asynchronous Connection Manager* (ACM) and *Network Monitor*. Also, it includes per-app CO DNS cache and Connection Pool cache which store, respectively, the history of the resolved domain names and the history of the connected hosts.

ADR performs DNS pre-fetching, asynchronous DNS resolution and provides cached DNS responses to the applications, based on the network stack capabilities. It has access to CO DNS cache for storing the pre-fetched app-specific DNS responses. ACM is responsible for establishing TCP connections and determining the best TCP connection between them at any moment. It utilizes Connection Pool cache for maintaining the pre-connected connection descriptors. Also, ACM interacts with CO DNS cache to fetch IP addresses for establishing connections. Finally, Network Monitor monitors the network conditions, e.g., analyzes the DNS responses and server capabilities, to optimize ADR and ACM operations.

B. OVERALL OPERATIONS

Fig. 6 shows the overall operations of NexGenCO. To perform asynchronous operations, NexGenCO creates two *Operation Threads* dedicated to IPv6 and IPv4 socket management separately. ADR and ACM manage the *Operation Threads* for asynchronous DNS resolution and TCP connect respectively. NexGenCO is capable of capturing the app-specific query-pattern, i.e., frequently resolved domain names, according to the technique proposed in [16].

ADR is able to trigger a DNS resolution on both operation threads and in ① performs an asynchronous DNS lookup. Then, each thread separately stores the response in the CO DNS cache. When the app requests for a resolution in ③, ADR i) replies with the resolution available in the cache or ii) trigger a resolution if not available in cache. In the latter case, ADR will notify the app when both IPv4 and IPv6 resolutions are ready. In the former case, instead, if NexGenCO has a valid entry in the cache for the requested domain name, ADR quickly replies to the app and provides near-zero resolution delay. It is worth to note that, according to studies such as [17], ADR sets the cache entry lifetime to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

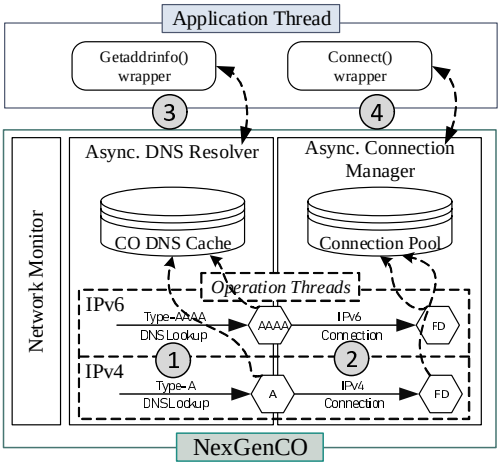

FIGURE 6. Overall operations of NexGenCO. NexGenCO creates two Operation Threads dedicated to IPv6 and IPv4 socket management separately. ADR and ACM manage the Operation Threads for asynchronous DNS resolution and TCP connect respectively.

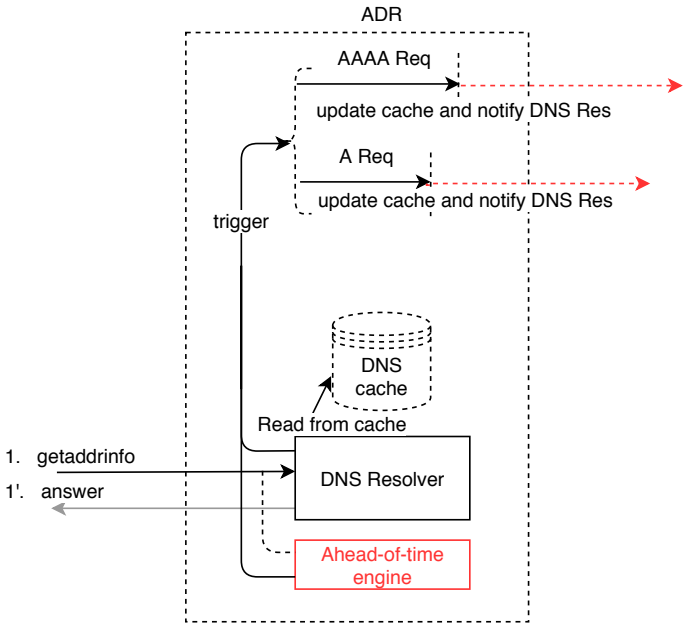

FIGURE 7. ADR and its concurrent operations and caching. The main goal of ADR is to reduce the DNS lookup time for an application. It incorporates i) asynchronous DNS resolution, ii) DNS caching and iii) ahead-of-time DNS lookup.

3600 seconds intending to reduce the DNS traffic generated from mobile devices and to reduce the resolution delay.

ACM, on each operation thread separately, waits for a domain name resolution. When ADR notifies a resolved domain name, ACM will trigger TCP connections on the corresponding Operation Thread, as shown in ②. Then, each thread stores the connections descriptors in the Connection Pool cache. When the app requests for a TCP connection ④, ACM i) replies with the best available connection, or ii) triggers a new connection. In the latter case, NexGenCO provides the descriptor to the app. In the former case, NexGenCO estimates the best available TCP connection, e.g., connection with minimum initial RTT, and it provides the descriptor to the app. Note that, even an in-progress connection, i.e., the thread opens the connection but the ACK from the server is not received yet, can be passed to the app in this case. In both cases, ACM assigns a connection file descriptor and closes the other connection or the other connection attempt.

Finally, the Network Monitor interacts with ADR and ACM. It is in charge to avoid resource wastage in NextGenCO. Hence, it monitors the connectivity availability and in case of unavailability, e.g., of IPv6 connectivity, suspends the corresponding thread till the connectivity comes back. Also, it monitors the network condition and in case of persistent failures, e.g., three subsequent failures, it suspends the thread.

C. ASYNCHRONOUS DNS RESOLVER (ADR)

Fig. 7 shows the inner details of ADR. The main goal of ADR is to reduce the DNS lookup time for an application. In general, a dual-stacked host performs both Type-`AAAA` (IPv6) and Type-`A` (IPv4) DNS resolution sequentially, which results in at least two RTT delay for an application. ADR reduces the DNS lookup time overhead and overcomes the limitations of Android DNS Cache as mentioned in Section.II. It incorporates i) asynchronous DNS resolution, ii) intelligent DNS caching and iii) ahead-of-time DNS lookup.

The Ahead-of-time engine is an important component of ADR. It triggers DNS resolutions before the application requests for it. Ahead-of-time engine continuously monitors all the outgoing DNS queries and recognizes the frequently triggered queries based on the number of DNS queries per web domain. During the application launch, these frequently-triggered DNS queries are used to pre-resolve the domain names before application requests for it.

When the app makes DNS request, DNS Resolver reads from cache and immediately replies to the app if a valid resolution is available in cache. If no valid resolutions are available in cache, ADR starts a DNS resolution by triggering DNS queries on different operation threads, which are dedicated for IPv6 and IPv4 socket management, concurrently. Then, each thread updates the DNS responses on the DNS cache and informs the DNS resolver. DNS resolver sorts the resolved IP addresses based on Destination Address Selection Rule, e.g., first address provided by DNS resolution. Then, DNS resolver replies to the app and provides the sorted resolved addresses.

Thus, NexGenCO is able to provide zero RTT DNS responses for an application by performing DNS resolution ahead of time and providing resolution in cache.

D. ASYNCHRONOUS CONNECTION MANAGER (ACM)

Fig. 8 shows the inner details of ACM. The primary objective of ACM is to reduce the TCP connection time for an application. It creates one or more parallel connections based on the sorted DNS responses. Whenever app requests for a connec-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

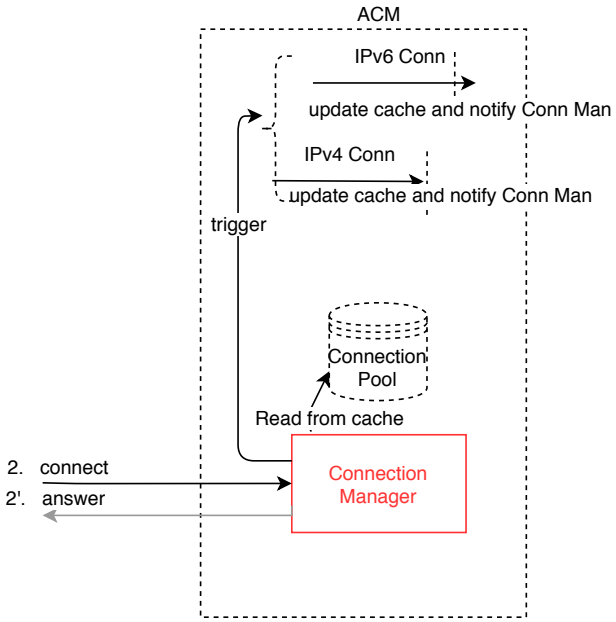

FIGURE 8. ACM and its concurrent operations and caching. The primary objective of ACM is to reduce the TCP connection time for an application. The main features of ACM are i) concurrent TCP connection attempts and ii) best path estimation.

tion, ACM estimates the best TCP connection available and maps it to the application descriptor for reducing the socket set-up delay. The main features of ACM are i) concurrent TCP connection attempts and ii) best path estimation.

When notified from ADR, ACM uses the sorted IP addresses and establishes TCP connections with IPv6 and IPv4 servers separately on each Operation Thread. As shown in Fig. 8, both connection attempts are made on the dedicated IPv6 and IPv4 Operation Thread simultaneously. After creating the TCP connections, ACM stores the descriptors in Connection Pool. When the application requests for a TCP connection, the Connection Manager of ACM estimates the best communication path between IPv6 and IPv4 connections. It considers the time taken for TCP three-way handshake, connection failure history and IP family of the TCP connection to choose the best connection at the moment. The idea is to select the connection with minimum initial RTT, no history of connection failures and also matching the app-requested IP family. Algorithm 1 provides a brief explanation of the best path estimation logic.

IV. MODEL FOR PERFORMANCE ANALYSIS

This section defines the mathematical model used to analyze the results created during our experimental analysis. Several rounds of tests were conducted to compare the NexGenCO performance with default Android and Happy Eyeballs algorithm. Table.1 explains the important terminologies used further in the performance model.

Algorithm 1 Best Path Estimation Logic

```
function EstimateBestPath(ipv4, ipv6)
    if (ipv6 == READY) && (ipv4 == READY) then
        return Lowest Initial RTT connection
    else if (ipv6 == READY) then
        return ipv6 connection
    else if (ipv4 == READY) then
        return ipv4 connection
    else
        return App requested IP-version connection
    end if
end function
```

TABLE 1. Terminologies used in performance modelling

Term	Description
`ANDCO`	Base Android connectivity mechanism
`iHECO`	Happy Eyeballs - iOS equivalent connectivity mechanism
`domain`	A domain name for which DNS lookups and TCP connections are performed by the application
`DNS lookup time`	The amount of time taken to complete the DNS resolution
`TCP connection time`	The amount of time taken to finish the TCP 3-way handshake
`Connectivity Overhead`	Total time taken for DNS lookup and TCP connection establishment
`Performance Gain`	The percentage of connectivity overhead reduced

A. DNS LOOKUP TIME ANALYSIS

Let u represent a `domain` used by the application. Consider that the network as well as device both supports dual-stack connections, then, the DNS lookup time for a `domain` u for default Android algorithm can be defined by following equation.

$$t_{dns}^{ANDCO}(u) = R_{AAAA}(u) + R_A(u) \tag{1}$$

Where, $t_{dns}^{ANDCO}(u)$ is the DNS lookup time for a `domain` u, $R_{AAAA}(u)$ is the actual RTT of Type-`AAAA` DNS lookup and $R_A(u)$ is the actual RTT of Type-`A` DNS lookup of `domain` u.

NexGenCO performs Type-`AAAA` and Type-`A` DNS resolution concurrently. Hence, DNS lookup time $t_{dns}^{NGCO}(u)$ for NexGenCO algorithm for a `domain` u can be represented by the following equation

$$t_{dns}^{NGCO}(u) = MAX\Big(R_{AAAA}(u), R_A(u)\Big) \tag{2}$$

Similarly, default connectivity mechanism in iOS, Happy Eyeballs algorithm also performs Type-`AAAA` and Type-`A` DNS resolution concurrently, but, it adds an additional resolution delay rd (refer to Fig.4) to prioritize IPv6 connections.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

Hence, DNS lookup time for HE algorithm $t_{dns}^{iHECO}(u)$ can be given by following equation.

$$t_{dns}^{iHECO}(u) = MAX\Big(R_{AAAA}(u), R_A(u) + rd\Big) \quad (3)$$

Where, rd is the resolution delay for Type-A DNS lookup.

B. TCP CONNECTION TIME ANALYSIS

Let us define the TCP connection time for an IPv4 server is $t_{v4}(u)$ and for a IPv6 server is $t_{v6}(u)$. So, the TCP connection time for each algorithm can be defined as follows. When the network and device support dual-stack connections, the default Android algorithm prefers the IPv6 connection

$$t_{tcp}^{ANDCO}(u) = \begin{cases} t_{v6}(u), & \text{if SUCCESS} \\ t_{v4}(u) + t_{v6}(u), & \text{otherwise} \end{cases} \quad (4)$$

Instead, NexGenCO always prefers the connection which has less initial RTT (along with the history of connection failures)

$$t_{tcp}^{NGCO}(u) = \begin{cases} t_{v6}(u), & \text{if } t_{v6}(u) < t_{v4}(u) \\ t_{v4}(u), & \text{otherwise} \end{cases} \quad (5)$$

Finally, HE prefers IPv6 connection ahead of IPv4 connection.

$$t_{tcp}^{iHECO}(u) = \begin{cases} t_{v6}(u), & \text{if } t_{v6}(u) < t_{v4} + \texttt{cad} \\ t_{v4}(u) + \texttt{cad}, & \text{otherwise} \end{cases} \quad (6)$$

Where, t_{tcp}^{ANDCO}, t_{tcp}^{NGCO} and t_{tcp}^{iHECO} are the TCP connection time of default Android, NexGenCO and HE respectively.

C. THEORETICAL PERFORMANCE GAIN

The overall performance gain of NexGenCO is the difference of its DNS lookup and TCP connection time with the default Android algorithm. Hence, the performance gain for any `domain` u can be defined as follows.

$$\tau^{NGCO}(u) = \Big(t_{dns}^{ANDCO}(u) - t_{dns}^{NGCO}(u)\Big) + \Big(t_{tcp}^{ANDCO}(u) - t_{tcp}^{NGCO}(u)\Big) \quad (7)$$

Where, $\tau^{NGCO}(u)$ is the performance gain of NexGenCO for `domain` u.

Similarly, the percentage gain for NexGenCO can be defined as follows.

$$\gamma^{NGCO}(u) = \frac{\tau^{NGCO}(u)}{t_{dns}^{ANDCO}(u) + t_{tcp}^{ANDCO}(u)} \times 100 \quad (8)$$

Where, $\gamma^{NGCO}(u)$ is the percentage gain of NexGenCO for `domain` u.

TABLE 2. Key parameters of Automated Test Framework

Parameter	Value
`Tool`	`pageloadtest` automated test framework
`Network`	Reliance Jio LTE Network
`LTE Band`	Band-40
`Data Source`	From Top 300 dual-stack websites by ALEXA ranking
`Duration`	30 days
`Frequency`	100-200 page loading per day
`No. of tests`	6000 cycles for 300 websites

V. PERFORMANCE EVALUATION IN CONTROLLED ENVIRONMENT

In this section, we present the experimental framework designed to compare the performance of NexGenCO against default Android and Happy Eyeballs version 2 (as it is the default connectivity mechanism in iOS). Also, the model presented in Section IV is verified by showing the adherence of obtained results to the ones expected by the model. For that, we implemented an automated test framework named `pageloadtest` and collected the results from the test setup for further analysis.

A. EXPERIMENT SETUP

We evaluated the performance of NexGenCO by comparing it with the existing algorithms of Android OS and also with IETF's Happy Eyeballs version 2. As depicted in Fig. 9, `pageloadtest` automated test framework is used to execute all three algorithms simultaneously to compare the connectivity overheads. We measured the *page loading time* of the top 10K websites featured by ALEXA ranking [4] for all three algorithms. For better evaluation, the `pageloadtest` filtered out the websites that were partially loaded or unable to load from the list of top sites. The filtering of these websites was done based on a fixed threshold value for DNS lookup and TCP connection time. Further, it selected only those websites which supported dual-stack connections. This automated test framework is compiled for the Android platform and deployed in Samsung smartphones.

The devices were connected to the Internet using Reliance Jio LTE network operator, which has a high IPv6 adoption rate of 93.66% [18]. For `pageloadtest` automated testing, we used three identical devices configured with ANDCO, iHECO, and NexGenCO algorithms respectively. Android webview component is used to build and run the `pageloadtest` framework to load the selected websites serially. These websites may contain more than one `domain` (as defined in Table.1) on the web page. `pageloadtest` collects network traces for all `domains` and Data Analyzer examines the DNS lookup and TCP connection time of each `domain`, as shown in Fig. 9, to generate an elaborated report. Table 2 summarizes the key parameters of the automated test campaign.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

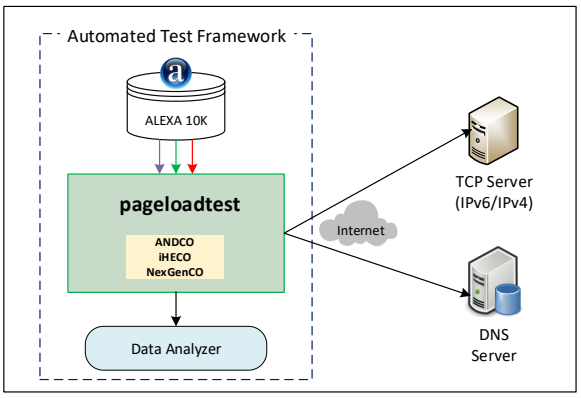

FIGURE 9. Automated Test Framework to evaluate NexGenCO. `pageloadtest` automated test framework is used to execute all three algorithms simultaneously to compare the connectivity overheads.

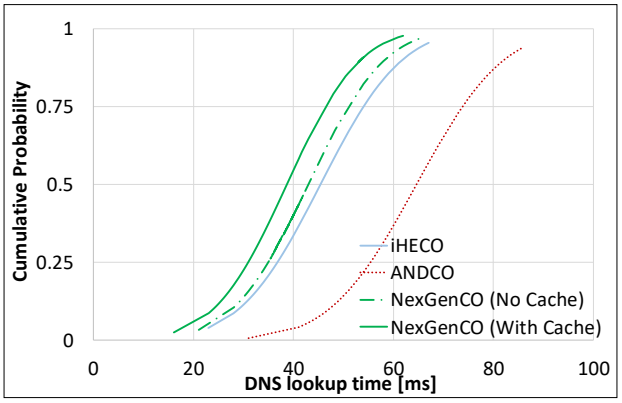

FIGURE 10. CDF of DNS lookup time. NexGenCO (t_{dns}^{NGCO}) reduces the DNS lookup time significantly.

B. RESULTS ANALYSIS

We analyzed the DNS lookup time of our experiments. Fig. 10 shows the CDF of DNS lookup time for the `domains` in the selected web pages with NexGenCO, iHECO, and ANDCO. As shown in the figure, NexGenCO (t_{dns}^{NGCO}) reduces the DNS lookup time using concurrent DNS resolution without causing any additional delay. Even without the DNS caching mechanism, NexGenCO performs better than Happy Eyeballs, as it does not favor any specific IP version. Whereas Happy Eyeballs (t_{dns}^{iHECO}) uses concurrent DNS lookup with a resolution delay (rd), preferring IPv6, the values are comparable but higher than NexGenCO. Since the Android platform performs the DNS resolution sequentially, the time taken for DNS lookup (t_{dns}^{ANDCO}) is about two times t_{dns}^{NGCO}. The result shows that 75% of DNS resolutions take less than 48 ms using the NexGenCO algorithm, whereas iHECO and ANDCO take 53 ms and 75 ms respectively to resolve a domain. Hence, we can observe around a 35% reduction in DNS lookup time for a `domain` supporting dual-stack network.

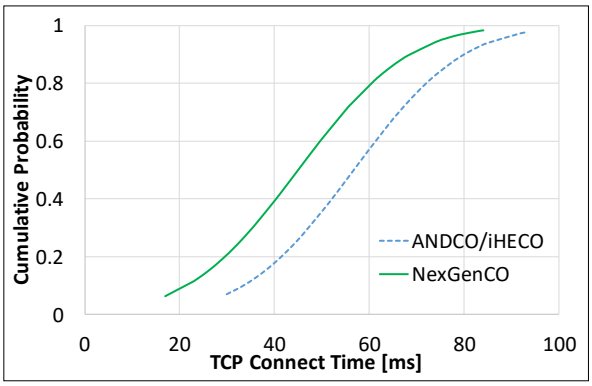

FIGURE 11. CDF of TCP connect time. NexGenCO reduces the TCP connect time by connecting to the server as soon as the DNS resolution is completed and mapping the best available connection.

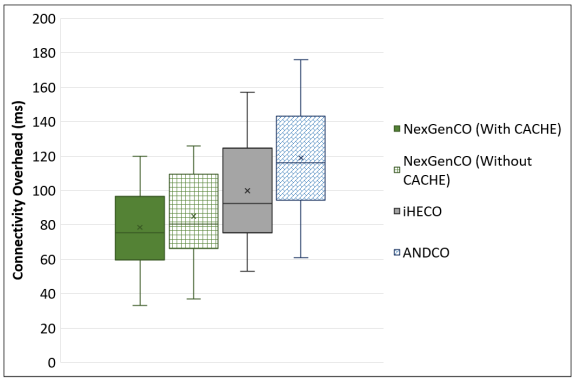

FIGURE 12. Connectivity Overhead Comparison. NexGenCO reduces the connectivity overhead of 30% and 35% with DNS cache and without respectively if compared with default Android.

Then, we analyzed the TCP connection establishment time of our experiments. Fig. 11 shows the CDF of TCP connect time for the `domains` in the selected web pages with NexGenCO, iHECO, and ANDCO. NexGenCO reduces the TCP connect time by connecting to the server as soon as the DNS resolution is completed and mapping the best available connection (IPv4/IPv6 connection) to the application when requested. Instead, ANDCO and iHECO create the connection upon receiving the application's request and always favor IPv6. Hence, both achieve higher TCP connection delays. More specifically, to connect with a TCP server, 75% of connections take less than 57 ms using the NexGenCO algorithm, whereas ANDCO and iHECO take 69 ms. Hence, we observe around an 18% reduction in TCP connect time for a `domain` supporting dual-stack network.

Finally, we analyzed the total connectivity overhead, which is the sum of DNS resolution delay and TCP connection delay. Fig.12 depicts the total connectivity overhead of the three algorithms using a box plot. For getting a complete result, we also evaluated the connectivity overhead of NexGenCO without its DNS caching mechanism. NexGenCO reduces connectivity overhead significantly, providing almost

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

zero RTT connections to the applications. We observe a reduction in the connectivity overhead of 30% and 35% by NexGenCO with DNS cache and without respectively if compared with ANDCO. Also, we perceive an improvement of about 15% and 20% in by NexGenCO with DNS cache and without respectively, if compared with iHECO. Indeed, the results show the median connectivity overhead value of 77 ms in NexGenCO with DNS cache, 81 ms in NexGenCO without DNS cache,117 ms in ANDCO, and 95 ms in iHECO.

According to results analysis and Equations 1 to 5, the relation between connectivity overheads of three algorithms can be represented as follows.

$$\begin{aligned}\left[t_{dns}^{NGCO} + t_{tcp}^{NGCO}\right] &< \left[t_{dns}^{iHECO} + t_{tcp}^{iHECO}\right] \\ &< \left[t_{dns}^{ANDCO} + t_{tcp}^{ANDCO}\right]\end{aligned} \tag{9}$$

Therefore, NexGenCO outperforms both Happy Eyeballs and default Android algorithms.

Eventually, it is worth to note that during the `pageloadtest` analysis, Monsoon Power Monitor Tool is utilized to estimate the power usage with Android platform, NexGenCO, and Happy Eyeballs. The results show no significant variations in the device power usage during our tests.

VI. LIVE-AIR PERFORMANCE EVALUATION

To understand the impacts and benefits of NexGenCO on end-user experience, we extended the evaluation to live-air network with popular Android applications.

A. EXPERIMENT SETUP

The proposed solution is successfully implemented in various Samsung models such as S10, S10+, A9 and A7 with Android Pie. We used two identical devices simultaneously for testing with and without NexGenCO solution and evaluated the solution by comparing different network parameters. Both devices were configured with the same hardware, software and network conditions. Experiments were conducted in both Wi-Fi and 4G/LTE networks. The results are taken from a live network with normal usage pattern for analyzing the real-time effects on end-users.

B. RESULTS ANALYSIS

Firstly, we analyzed the performance of DNS resolution in Android with NexGenCO in terms of the number of queries generated and the DNS lookup time of several popular applications. The trace I/O graph in Fig.13 shows the average DNS traffic for the first three minutes after launching an app with NexGenCO and default Android. NexGenCO performs concurrent DNS resolution ahead of time for minimizing DNS lookup time. As shown in the graph, NexGenCO's asynchronous DNS pre-fetching results in higher DNS traffic during the app startup. Later, due to DNS caching, it drastically reduces the number of DNS lookup requests. Thus, by average, NexGenCO minimizes overall DNS traffic from

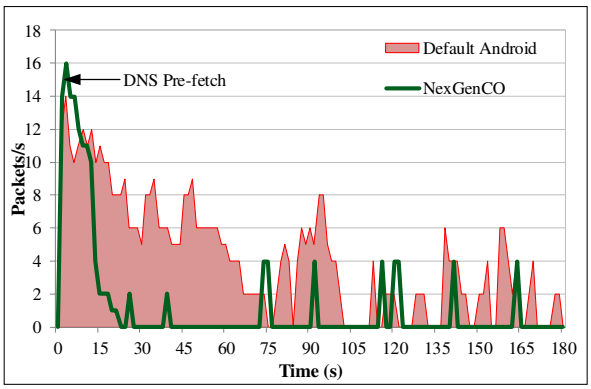

FIGURE 13. DNS traffic generated by an app in Android with and w/o NexGenCO. NexGenCO performs concurrent DNS resolution ahead of time for minimizing DNS lookup time. NexGenCO's asynchronous DNS pre-fetching results in higher DNS traffic during the app startup. Later, due to DNS caching, it drastically reduces the number of DNS lookup requests.

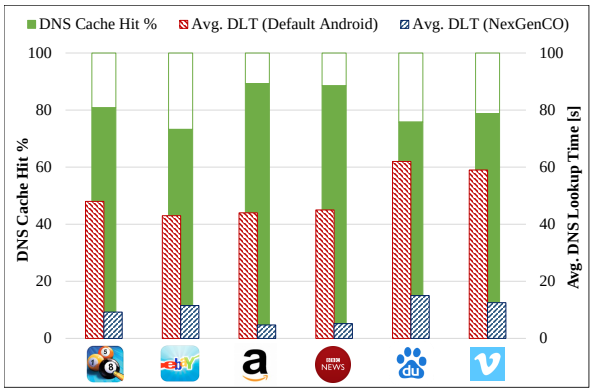

FIGURE 14. DNS cache hit percentage and DNS Lookup Time (DLT) in popular Android apps with and w/o NexGenCO. NexGenCO minimizes the average DLT by up to 89%.

52.81% to 78.22%. To further evaluate the effect of DNS cache on Android apps, Fig. 14 depicts the DNS cache hit rate of NexGenCO. As a result of the intelligent caching mechanism, on an average, NexGenCO DNS cache achieves 81.07% hit rate. Fig. 14 further compares the average DNS lookup time (DLT) with NexGenCO and default Android for various popular apps. NexGenCO minimizes the average DLT by up to 89% consistently.

Then, we evaluate the impact of TCP connection time in NexGenCO performance. NexGenCO realizes TCP preconnect ahead of the application request. Since NexGenCO avoids the TCP three-way handshake overhead by making concurrent TCP connections attempts, it provides zero RTT overhead TCP connections to the applications. Fig. 15 reports the average number of TCP connections established by the apps and TCP pre-connections created by NexGenCO along with zero RTT overhead percentage. Also, Fig. 16 summarizes the average TCP connection time improvement using NexGenCO. NexGenCO provides from 35.71% to 65.11% zero RTT connections and reduces the effective TCP connec-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3014389, IEEE Access

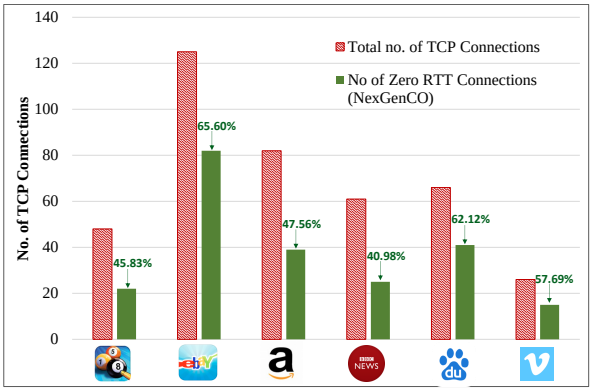

FIGURE 15. Number of TCP connections opened in Android popular apps and percentage of zero RTT TCP connections obtained by means NexGenCO. NexGenCO provides from 35.71% to 65.11% zero RTT connections.

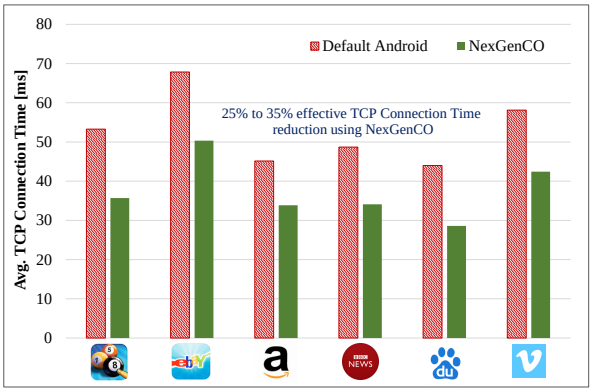

FIGURE 16. TCP connection time in Android popular apps with and w/o NexGenCO. NexGenCO reduces the effective TCP connection time from 25% to 35%.

tion time from 25% to 35%.

Finally, we estimate the performance of NexGenCO in terms of browser page loading time. We developed a test application that loads webpages one by one (from the list provided) and provides the PLT for each page. Fig.17 shows the impacts of page loading time of the commonly used 100 websites from Alexa ranking. We chose popular websites with page loading time lie in between 1500 ms to 3000 ms. NexGenCO significantly improves page loading time by 12.81% to 18.33% consistently.

The results using `pageloadtest` (controlled environment) and live-air networks confirm that NexGenCO minimizes connectivity overhead of protocol stack and improves the end-user experience significantly.

VII. CONCLUSION

As 5G and IoT are emerging and IPv6 adoption rate is booming, the need for efficiently handling the connection overhead caused by dual-stack network is inevitable. In this paper, we propose a novel solution named NexGenCO to improve the dual-stack connectivity and support intelligent DNS caching. It is a client only software solution for minimizing latency

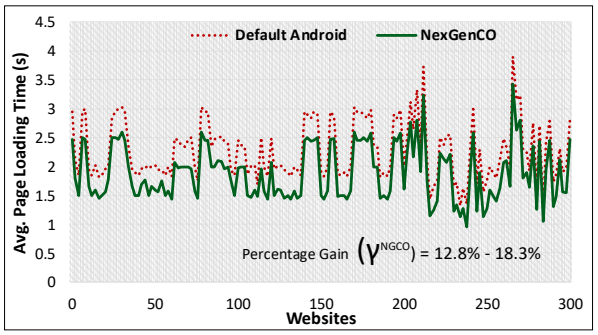

FIGURE 17. Page Loading Time of ALEXA popular wbsites. NexGenCO significantly improves page loading time by 12.81% to 18.33%.

caused by the network protocols, which is easily deployable in Android based devices. It is a user-space solution and does not require any changes in the kernel, existing network protocols, middle boxes or servers. NexGenCO is a light-weight solution which is prototyped in Samsung devices with Android Pie and evaluated its performance with multiple defined scenarios. Extensive analysis was conducted using `pageloadtest` framework. NexGenCO significantly reduced network protocols latencies without causing any performance degradation. It improved the page loading time up to 18% consistently.

In addition to finding the best TCP communication path among IPv4 and IPv6 servers, exploring all available content server path quality is a area of interest for us. Hence, we extend our scope of work to find the best communication path among all the content servers with zero connectivity overhead to the application.

REFERENCES

[1] Statista. Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions). (Accessed on 09-Jan-2020). [Online]. Available: https://www.statista.com/statistics/471264

[2] GoogleIPv6. IPv6 Adoption Statistics. (Accessed on 09-Jan-2020). [Online]. Available: https://www.google.com/intl/en/ipv6/statistics.html

[3] InternetSociety. World IPv6 Launch. (2012). Retrieved Dec 04, 2019 from http://www.worldipv6launch.org.

[4] "Alexa.com Website Traffic Statistics," https://www.alexa.com/, accessed: 2019-07-21.

[5] "Dan Wing - AAAA and IPv6 Connectivity Statistics," http://www.employees.org/ dwing/aaaa-stats/, (Accessed on 09-Jan-2020).

[6] J. Pickard, M. Angolia, and D. Drummond, "Ipv6 diffusion milestones: Assessing the quantity and quality of adoption," *Journal of International Technology and Information Management*, vol. 28, no. 1, pp. 2–28, 2019.

[7] D. Schinazi and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency," RFC 8305, Dec. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8305

[8] D. S. Sabareesh, G. V. P. Reddy, S. Jaiswal, J. M. Ppallan, K. Arunachalam, and Y. Wu, "Redundant TCP connector (RTC) for improving the performance of mobile devices," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2019, pp. 1–7.

[9] K. Arunachalam, J. M. Ppallan, S. Jaiswal, R. S. Lingappa, V. Balasubramanian, and K. Subramaniam, "Layer 4 Accelerator (L4A) for optimizing network protocol latencies in mobile devices," in *2018 IEEE 20th International Conference on High Performance Computing and Communications (HPCC)*, June 2018, pp. 439–448.

[10] M. Almeida, A. Finamore, D. Perino, N. Vallina-Rodriguez, and M. Varvello, "Dissecting dns stakeholders in mobile networks," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17, 2017, pp. 28–34. [Online]. Available: http://doi.acm.org/10.1145/3143361.3143375

[11] D. Wing and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts," RFC 6555, Tech. Rep. 6555, Apr. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6555.txt

[12] V. Bajpai and J. Schönwälder, "A longitudinal view of dual-stacked website: Failures, latency and happy eyeballs," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, Apr. 2019.

[13] Google. DNS Prefetch Control. (Accessed on 09-Jan-2020). [Online]. Available: https://dev.chromium.org/developers /design-documents/dns-prefetching

[14] D. Ghosh, M. Bapst, C. Lott, R. Attar, G. Cherian, L. He, and D. Garg, "Adaptive DNS pre-fetching," Feb. 4 2014, uS Patent 8,645,501. [Online]. Available: https://www.google.com/patents/US8645501

[15] J. Roskind, "Reduction in redirect navigation latency via speculative preconnection," Dec. 2 2014, uS Patent 8,903,946. [Online]. Available: https://www.google.com/patents/US8903946

[16] J. M. Ppallan, K. Arunachalam, S. Jaiswal, D. S. Sabareesh, S. Seo, and M. R. Kanagarathinam, "Flare-DNS Resolver (FDR) for optimizing DNS lookup overhead in mobile devices," in *2019 IEEE Consumer Communications and Networking Conference (CCNC)*, January 2019, pp. 1–7.

[17] G. C. M. Moura, J. Heidemann, R. d. O. Schmidt, and W. Hardaker, "Cache me if you can: Effects of dns time-to-live," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. New York, NY, USA: ACM, 2019, pp. 101–115. [Online]. Available: http://doi.acm.org/10.1145/3355369.3355568

[18] "APNIC Labs : IPv6 Capable Rate by country," https://stats.labs.apnic.net/ipv6/IN, accessed: 2019-07-21.

JAMSHEED MANJA PPALLAN (M'17) is currently working as a Research Engineer for Samsung R&D Institute India-Bangalore. Previously, he worked as a Senior Software Engineer for Huawei Technologies India Private Limited and as Associate Programmer for National Informatics Centre, Govt. of India. He received a B.Tech degree in computer science from Cochin University of Science and Technology, Kerala, India, in 2012. He has 8+ years of industry experience in software research and development. His research interests include next-generation transport layer protocols, cross-layer optimization, smartphone operating system, and green communication.

SWETA JAISWAL (M'20) has 11+ years of experience in software research and development in the telecommunication industry. Currently, she is working as Chief Engineer for Samsung R&D Institute India, Bangalore. Previously she worked as a software engineer for Tata Consultancy Service Ltd. She received her B.Tech degree in electronics and communication from the Vellore Institute of Technology, Vellore, Tamil Nadu, India. Her research interest lies in Communication and Networking which include next-generation transport layer protocols, Multi-access Edge Computing (MEC), green communication techniques, and cross-layer optimization.

KARTHIKEYAN ARUNACHALAM (M'16, SM'19) has 15+ years of extensive research experience in Transport layer protocols. He is an IEEE senior member and ACM member. Currently, he is working as an Architect for Samsung Research institute India Bangalore. His current research interests include next-generation transport layer protocols, cross-layer communication, and mobile edge computing. Previously he worked as a senior associate in Novell Software Development (India) Private Limited, senior software Engineer in Huawei Technologies India Private Limited, and software engineer in Protechsoft Technologies Private Limited. He received a B. Tech degree in information technology from Anna University, Chennai, India.

PASQUALE IMPUTATO received the M.Sc. and Ph.D. degrees from the University of Napoli Federico II in 2015 and 2019, respectively. He is currently a research fellow at Department of Computer Engineering at the University of Napoli. He was a visiting researcher at the Centre Tecnològic de Telecomunicacions de Catalunya (2017-2018). His research interests include wireless networks and the bufferbloat problem.

STEFANO AVALLONE received the M.Sc. and Ph.D. degrees from the University of Napoli Federico II in 2001 and 2005, respectively. He is currently an Associate Professor with the Department of Computer Engineering at the University of Napoli. He was a visiting researcher at the Delft University of Technology (2003-04) and at the Georgia Institute of Technology (2005). He is on the editorial board of Elsevier Ad Hoc Networks and the technical committee of Elsevier Computer Communications. His research interests include wireless mesh networks, 4G/5G networks and the bufferbloat problem.

DRONAMRAJU SIVA SABAREESH (M'19) is graduated from JNTU Kakinada in Electronics and Communications Engineering and received his Post Graduation degree from IIT Kharagpur in Telecommunication Systems Engineering. Currently, he is working as a Chief Engineer at Samsung R&D Institute, Bangalore. He has 7+ years of work experience in the Android Telephony Framework, Android Connectivity Framework, Android Application Development, MP-TCP, Radio Interface Layer (RIL), Design and Development of Transport Layer Protocols. His current research interests are in Communication & Network which include next-generation transport layer protocols, Mobile Edge Computing (MEC), cross-layer optimization, and green communications.

MADHAN RAJ KANAGARATHINAM (M'18, SM'20) received the BE degree in computer science and engineering from Anna University, Chennai, India, in 2012. He has seven years of working experience in design and development of TCP/IP protocols, Multi-path TCP and UNIX flavored operating systems. Currently, he is working as a Chief Engineer for Samsung R&D Institute India Bangalore. Previously, he worked as an Engineer with Aricent Technology (India) Private Limited. He is the author of 15 articles, and more than 20 inventions. His current research interests include communication and network which include Pre-6G/Beyond 5G, Next Generation Mobile Network, software-defined network architecture, transport layer protocols, and cross-layer optimization technique. He is a senior member of the IEEE, and a member of the ACM.