Enabling Context-aware HTTP with Mobile Edge Hint

Peng Qian, Ning Wang, Gerry Foster, Rahim Tafazolli
Institute for Communication Systems, University of Surrey,
Guildford GU2 7XH, UK. Email: {p.qian, n.wang, g.foster, r.tafazolli}@surrey.ac.uk

Abstract—**Due to dynamic wireless network conditions and heterogeneous mobile web content complexities, web-based content services in mobile network environments always suffer from long loading time. The new HTTP/2.0 protocol only adopts one single TCP connection, but recent research reveals that in real mobile environments, web downloading using single connection will experience long idle time and low bandwidth utilization, in particular with dynamic network conditions and web page characteristics. In this paper, by leveraging the Mobile Edge Computing (MEC) technique, we present the framework of Mobile Edge Hint (MEH), in order to enhance mobile web downloading performances. Specifically, the mobile edge collects and caches the meta-data of frequently visited web pages and also keeps monitoring the network conditions. Upon receiving requests on these popular webpages, the MEC server is able to hint back to the HTTP/2.0 clients on the optimized number of TCP connections that should be established for downloading the content. From the test results on real LTE testbed equipped with MEH, we observed up to 34.5% time reduction and in the median case the improvement is 20.5% compared to the plain over-the-top (OTT) HTTP/2.0 protocol.**

I. Introduction

Today, the HTTP-based web page content applications often suffer from poor web page loading performances in mobile network environments [1], [2]. In recent years, substantial efforts have been invested on improving the load time performance from different perspectives. On the one hand, the next generation of HTTP protocol 2.0 is able to tackle the existing limitations such as Head-of-Line (HOL) blocking at the application layer [3]. On the other hand, various HTTP proxy-based approaches break the end-to-end connection in order to enable web content pre-fetching, caching, offloading and compression to enhance user experiences [4]–[6]. Nevertheless, till now noticeable performance gains cannot always be achieved under different network conditions and web page characteristics [7], [8].

To overcome the inefficiency of repeating TCP handshaking and also the HOL issue, the HTTP/2.0 protocol multiplexes all the web content objects contained in a webpage to be delivered through one single TCP connection [3]. This single connection is expected to fast ramp up the available bandwidth because the initial congestion window of a TCP connection has been suggested to increase to 10 [9]. However, recent experiments have shown that under certain conditions HTTP/2.0 is only able to achieve marginal improvement when compared to its predecessor HTTP/1.0 [1], [7], [10]. Specifically, this single short-lived TCP connection may experience high idle or waiting time during the slow start phase [2], [11], and the complex object dependency within a page also aggravates the idle time [7], [12]. This idle time varies according to different content sizes, bandwidth and end-to-end delay [10], thus limiting the bandwidth utilization and affecting the page load time.

In addition to the over-the-top (OTT) HTTP/2.0 protocol evolution, another thread of research is towards intelligent mobile content access with the assistance of network functions, typically through web content proxies. Current proxy-based techniques applied in mobile network environments share two common features [4]–[6]: (1) Splitting end-to-end connections between wireless and wired parts, and (2) Employing various network operations such as content prefetching and caching for mobile users. However, again these widely deployed proxies are unable to comprehensively achieve improvements according to different context conditions [5], [8], [13]. In addition, splitting of end-to-end connections encounters issues like unexpected timeout [14] or marginal improvement [8] in real mobile networks.

Considering above challenges, we propose a new scheme of HTTP with context awareness based on the emerging Mobile Edge Computing (MEC) concept [15]. In the context of designing future 5G networks, MEC has been proposed as an advanced technique that allows smart network functions embedded at the mobile network edge for enhancing OTT application performances. One typical example on supporting adaptive video streaming applications is for the MEC server to provide just-in-time feedback on radio access network conditions towards the original video source in order for the latter to perform necessary adaptations [15]. Taking the MEC principle as the starting point, we aim to further tackle the issue of suboptimal web content downloading performances. The target is to reduce the page load time in mobile networks by taking into account specific context conditions, not only including dynamic network conditions but also distinct web page complexities. The proposed framework is called context aware HTTP with Mobile Edge Hint (MEH). Different from the majority of existing work where the mobile network proxy acts as a man-in-the-middle in terms of web content acceleration [4], [6], [16] , our approach strives to maintain end-to-end connectivity where necessary (especially for in-band downloading of web objects), while having the mobile edge to provide necessary hint back to the clients. Based on the hinted information, a client will execute a local algorithm to select an optimal connection number to overcome the idle time for this specific page complexity and current network condition, thus leading towards reduced page downloading time. Note that different form long-lived application likes video streaming, the variation of network condition during a single page loading is minor because the short-lived web page loading only lasts for very few seconds. Therefore the edge hint and the client-side algorithm only need to be executed once at the starting phase when user requests to a new page.

In the rest of the paper, we will discuss on the following topics. In section II, we present the detailed design of the MEH framework. Based on this, a numerical study of the impact of webpage complexity and network conditions will be elaborated in section III, followed by a plug-and-play algorithm which determines the optimized number of parallel end- to-end connections against dynamic conditions. In section IV, we describe the prototype implementation of the proposed MEH framework, based on which we present our final results obtained from experiments in a real LTE network. Then we conclude our work in section V.

II. Design of Mobile Edge Hint framework

In this section we first describe our basic design rationale by briefly explaining why webpage downloading performance can be enhanced through adaptive number of parallel TCP connections according to specific webpage complexities and network conditions. Then we formally introduce the MEH framework together with the necessary signaling mechanism to enable web clients to select the optimized number of connections according to the hinted context information from the mobile edge. Such a framework will be further detailed according to the mobile edge side and the client side, and finally in this section we discuss some practicality issues relating to MEH.

Adaptive Parallel Connections
As previously mentioned, HTTP/2.0 only adopts one single TCP connection. However, this single TCP connection experiences idle time during its slow start phase and also leads to low bandwidth utilization in mobile networks [2], [11]. Increasing the initial congestion window is a typical way to overcome such an idle time issue. The evaluations in [17], [18] reveal that for different network Bandwidth-Delay Production (BDP) and webpage content sizes, the effect originating from increasing congestion window can be significantly different. That is to say, depending on the characteristics of the webpage being requested and also the current network conditions including latency and bandwidth availability, it is beneficial to launch different number of TCP connections in downloading all the content objects contained in a webpage. To this end, we design an algorithm at client side to adaptively select and open parallel connections to aggregate the initial congestion window size according to the aforementioned context information. This algorithm is able to identify the idle time of the target page under specific network condition and adaptively select an optimal connection number for webpage downloading operations. We leave the details and numerical analysis of this algorithm in section 3, but first focus on the general MEH framework with the focus on how the mobile edge is able to provide information hints on the necessary context information to the client side in order for the latter to make optimized decisions on the number of connections.

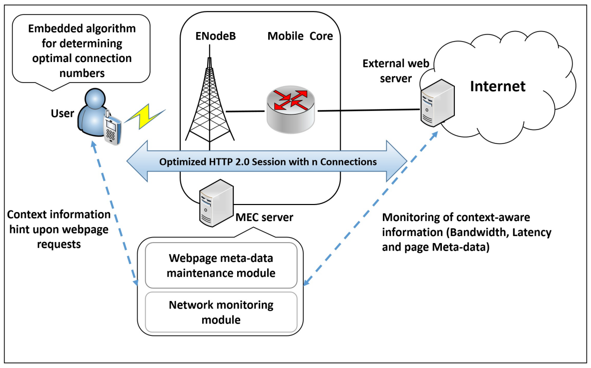
Embedded algorithm for determining optimal connection numbers
User
ENodeB
Mobile Core
External web server
Internet
Optimized HTTP 2.0 Session with n Connections
MEC server
Context information hint upon webpage requests
Monitoring of context-aware information (Bandwidth, Latency and page Meta-data)
Webpage meta-data maintenance module
Network monitoring module

Fig. 1. Overview of Mobile Edge Hint HTTP framework

Framework of HTTP/2.0 with Mobile Edge Hint:
The architecture of proposed framework is based on the MEC concept (see Fig. 1). Holding the optimization algorithm locally, a mobile client utilizes the monitoring and meta-data storage capability at the MEC server to obtain context-aware information as the input parameters to the optimization algorithm. Such inputs include: (1) page meta-data (embedded URLs and page size) and (2) network condition including the end-to-end latency and bandwidth availability. Conventionally, a client cannot understand the meta-data of the embedded content objects until they are fully downloaded and in general the webpages always have high variance on page sizes and embedded object sizes [19]. Similarly, to determine the end-to-end BDP associated with the path to external server, the challenge in today's mobile network is that the latency at wired part may dominant the end-to-end latency [20] and due to this, contacting the external network entity leads to a waste of time and jeopardizes the page loading time. To this end, the ability of direct network monitoring and maintenance of (relatively static) web content meta data at MEC server located at the mobile edge (e.g. eNodeB) can be utilized to provide necessary hints on the required context information on the client side. Moreover, due to its close proximity to mobile user, these metrics can be online delivered to mobile user through the query Application Program Interface (API) without incurring latency to external server.
We describe the detail of the intelligence at mobile edge and client side as follows:

Intelligence at mobile edge: Collection of content-aware information (offline) and fast hints (online)
The intelligence at the mobile edge includes both offline operation (meta-data trace and network monitoring), and online operations (fast hints back to mobile clients upon their webpage requests). (See Fig. 1 and the dash line in Fig. 2.)
(1) Capturing (popular) web content meta-data trace. The meta-data trace application periodically loads the target page, extracts the meta-data of each object from the response

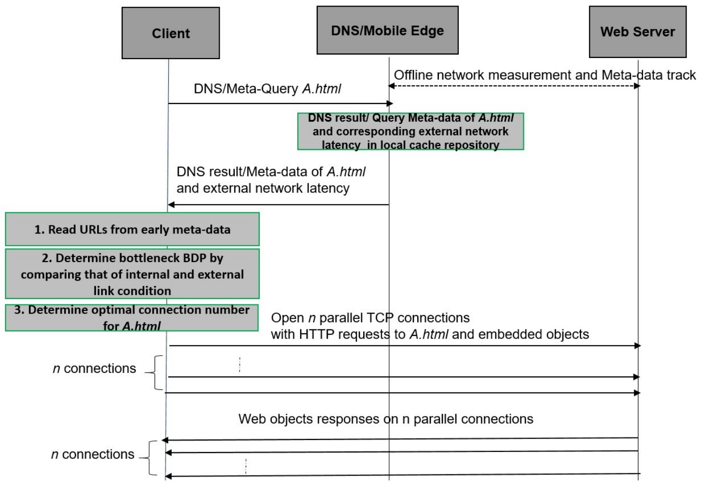
Client
DNS/Mobile Edge
Web Server
Offline network measurement and Meta-data track
DNS/Meta-Query A.html
DNS result/ Query Meta-data of A.html and corresponding external network latency in local cache repository
DNS result/Meta-data of A.html and external network latency
1. Read URLs from early meta-data
2. Determine bottleneck BDP by comparing that of internal and external link condition
3. Determine optimal connection number for A.html
Open n parallel TCP connections with HTTP requests to A.html and embedded objects
n connections
Web objects responses on n parallel connections
n connections

Fig. 2. Work process of the proposed framework

headers and caches them at local meta-data cache repository. This meta-data includes information of each embedded object, such as object size, object URL, object type, last update time etc. It is worth mentioning that for scalability purposes this operation only targets at popular webpages that are frequently visited by local mobile users, and such popularity can be monitored by the network through specific mechanisms [21].
(2) Network measurement service. The network measurement application also periodically measures the end-to-end latency from the base station to web server and caches them. We do not measure the bandwidth at wired network part because according to existing performance test in cellular network, the available bandwidth is still much smaller than that in wired part [2], [11].
(3) Fast mobile hints. Upon an incoming request for a popular webpage for which the corresponding meta-data has been maintained locally, the MEC server will be able to provide hints back to the client in order for the latter to make its own decisions on the total number of connections that should be established in order to optimize the downloading performance. Such hints effectively include latency at the wired part (based on network monitoring), and also the meta-data about the webpage being requested. In our framework, the MEC server also contains a local DNS component which has already been proposed in ETSI. Therefore the hint packets can be delivered along with the DNS response message through an API. Figure 2 shows the detailed operation procedure on how the MEC server gathers necessary context information and provides fast hints back to webpage content clients.

Intelligence at client side: Cooperating with mobile edge, the HTTP 2.0 client sends requests to all embedded URLs on parallel connections
The solid line in Fig. 2 shows the online work process at the client side. Before sending a request on a webpage, the client first sends a DNS query which can be handled by a local DNS function embedded inside the MEC server, and a meta-data query message is also sent along with this DNS query message. Upon receiving the query, the MEC server is able to immediately send back the meta-data of the webpage and the context information about the network condition. Upon receiving this edge hint, the client will execute its local optimization according to the following steps: (1) It reads and rebuilds the meta-data information of each embedded object in the webpage, checks its local cache information and compares with the meta-data in order to determine the URLs and size of pending page (obtained by adding up the size of each object) to be requested. (2) It measures the latency and bandwidth for the cellular wireless network part and binds these values with the external network condition (i.e. from the mobile edge to the external content server) in order to determine the BDP on the end-to-end path. As discussed before, since it is widely believed that the bandwidth at the wireless cellular link is always the bottleneck along the end-to-end path [2], in the rest of this paper, we assume the BDP equals to bandwidth in the wireless cellular link multiplied with the concatenation of wired and wireless latency. (3) It uses the end-to-end BDP and the pending page size as inputs, and executes the local optimization algorithm to determine the number of parallel connections to destination web server. Such an algorithm at the client side is effectively a plug-and-play function, and we will introduce one representative algorithm in the next section together with the analysis of the corresponding performance. Once the total number of connections is determined, the client will directly send object requests through these parallel connections. In addition, in order to keep the load balance between these parallel connections, since the client understands the size of each object, it can allocate objects with same aggregated size on each connection. Effectively, the page loading time experienced at the client side can be reduced because the network idle time during TCP slow start can be overcame by aggregating the congestion window of parallel TCP connections.

Practicality considerations
Considering the practical realization of MEH, existing works like [4]–[6] also leverage different measurement tools to trace the page meta-data or network condition as the input to their algorithms. We select the Chrome browser with *chrome-har-capturer* [22] tool to periodically load and trace the target web page. The HTTP response headers of each embedded object are local maintained to build up the meta-data cache of traced page. For other objects dynamically executed by user, we believe prior approaches [6], [23] have been able to address such issues, and therefore we directly borrow their methods in our framework. Note that there are also valuable work in the literature which achieves early object dependency awareness and corresponding object re-scheduling [23], [24], these approaches can be complementarily embedded in our framework in order to complementarily tackle the advanced object dependency issue. Regarding the network condition measurement, a Channel Quality Indicator (CQI) based approach proposed in [25] can be used to estimate available bandwidth at client side. The latency on the wireless cellular

part can be obtained by the exchanged query messages between client side and the MEC server.

III. Numerical analysis of HTTP/2.0 idle time

In this section, we give a numerical analysis of TCP idle time during a HTTP/2.0 page loading and introduce an adaptive parallel connection algorithm to determine the number of parallel connections.

A. *Impact of idle time under varying network condition and page size*

In HTTP/2.0, the idle time on the single TCP connection session consists of two parts: (1) the idle time caused by parsing of object dependency [7], [12] and (2) the idle time caused by TCP slow start [17], [26]. For the former, recent efforts have been invested to break the 'download-parse-download' dependency based on early-stage URL awareness [4], [27]. In this case, the early known URLs can be concurrently requested and downloaded on the TCP connection, thus the idle time caused by object dependency parsing can be minimal. To keep pace with these latest approaches, we also assume that this type of idle time can be ignored. In contrast, due to the increasing trend of web content size and rapid evolution of network condition, idle time caused by TCP slow start can frequently take place in LTE network for short-live application [2], [11], although the initial congestion window has already been to promoted to 10 [9], [18]. In the following parts, we mainly focus on the idle time caused by TCP slow start.

Considering the completion time model of this single TCP connection, existing works such as [17], [26], [28] provide sufficient works. The completion time T_{HTTP} can be obtained as:

$$T_{\text{HTTP}} = 2RTT + \frac{S_{\text{page}}}{\mu} + \gamma\left[RTT + \frac{S_{\text{mss}}}{\mu}\right] - 2(1.5^{\gamma} - 1)\frac{IW * S_{\text{mss}}}{\mu}$$

where

$$\gamma = min\left\{\left\lfloor log_{1.5}\left(1 + \frac{RTT}{2IW * S_{\text{mss}}/\mu}\right) + 1\right\rfloor, \left\lceil log_{1.5}\left(\frac{S_{\text{page}}}{IW * S_{\text{mss}}}\right) + 1\right\rceil - 1\right\}, \tag{1}$$

The notation γ denotes the number of rounds that the congestion window exponentially increases during the slow start phase, RTT refers to the round-trip time, IW is the initial congestion window which equals to 10, μ indicates the available bandwidth, S_{page} and S_{mss} are the size of a web page and a TCP packet, respectively. The constant 1.5 in this equation indicates that the delayed-ack is enabled and this is the default setting in today's Linux system.

We also follow the same assumptions in [17], [26], [28] that the exponentially increasing congestion window is not limited by the TCP *sshthreshold* and when the congestion window increases to full pipe size during slow start phase, the rest part of content will be transferred with the full rate μ.

To assess the effect of idle time, we first calculate T_{idle} (see in equation 2) and then focus on its proportion of the total completion time as $T_{\text{idle}}/T_{\text{HTTP}}$. We vary the page size from small to large by following the Fig. 3. Figure 3 shows the Cumulative Distribution Function (CDF) of size of top 200

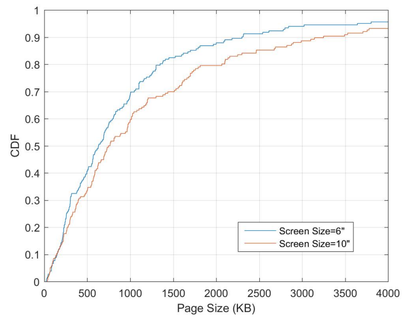

Fig. 3. Size of web page (mobile version, screen size 6″ and 10″)

websites in the Internet. Since our scenario is webpage downloading via mobile network infrastructures, the webpage size is measured from mobile version pages with the median value of page size is 756KB (screen size 10″) and 663KB (screen size 6″), respectively. These long-tailed curves also present higher average value (1.39MB and 1.08MB, respectively) due to the existence of some large pages which has no mobile version. We also vary the network condition by changing the BDP value from low to high. These values are referred from [2], [11], [20]. For simplicity, we fix bandwidth and change RTT to emulate the BDP range.

$$T_{\text{idle}} = \gamma\left[RTT + \frac{S_{\text{mss}}}{\mu}\right] - 2(1.5^{\gamma} - 1)\frac{IW * S_{\text{mss}}}{\mu} \tag{2}$$

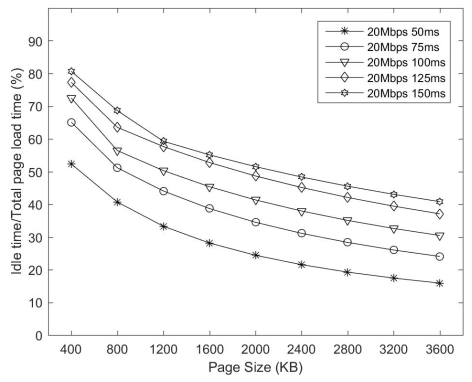

Fig. 4. Percentage of idle time in total page load time

Figure 4 depicts the percentage of idle time in total page loading time for varying varied page sizes and network conditions. In terms of different web pages, the increasing of page size leads to a reduction of the proportional idle time. For instance, when network BDP is low (e.g. with 20Mbps bandwidth and 50ms latency), a 400KBytes page suffered from 52.3% idle time while a 3600KBytes page only experiences 15.9% idle time. This is because the most part of large page are transmitted during full BDP period, which alleviates the effect of idle time. Similarly, regarding the impact of network conditions, large BDP extends the time that a connection stays at exponentially increasing period during the slow start phase.

Therefore a noticeable increment on idle time can be observed when latency increases from 50ms to 150ms. For example, the idle time rises from 52.3% to 80.1% for a 400KBytes page and for a web page which of 3600K Bytes size, this same trend can also be observed (its idle time increases from 15.9% to 40.8%). These above observations indicate that the idle time on one single TCP connection can significantly affect the page loading time under today's page load context, and this negative effect varies according to different page size and network conditions.

B. Connection selection algorithm

Since opening parallel connections is able to achieve higher congestion windows, the idle time can be reduced by the aggregated congestion windows according to [17]. Note that parallel TCP connections has already been investigated for aggregating more bandwidth, but our algorithm differs from them in the following aspects: (1) Some approaches only use fix number of TCP connection [16] which misses the consideration of content size and network condition. (2) For the approaches which use dynamic number of TCP connections [29], they only consider one of slow start or steady state phase. But as discussed before, takes into account today's mobile web page size, the TCP congestion window of mobile version web pages will first experience the exponential increasing period and then stays at the full BDP until the page completes.

Algorithm 1 Algorithm for select number of parallel connections n

```
$ConnectionNumber_{\max}$=min($ceil(BDP/(IW * MSS))$,$ceil(S_{page}/(IW * MSS)))$
$i = 2$, $C(i)$=1/$ConnectionNumber_{\max}$,
$G(i)$= ($T_{HTTP}(i-1)$-$T_{HTTP}(i)$)/$T_{HTTP}(1)$
while true do
    if $G(i) >= C(i)$ then
        i=i+1;
    else
        break;
    end if
end while
n = i-1;
```

In order to decide an optimized connection number, our algorithm evaluates the gain and cost of increasing each one connection. (See Algorithm 1). Considering a network with fixed BDP, for the page whose size is not less than the BDP, the maximum number of connections to fully utilize this given BDP pipe is fixed as $ConnectionNumber_{\max}$ = $ceil(BDP/(IW * MSS))$, or for page whose size is less than the BDP size, the maximum connection number equals $ConnectionNumber_{\max}$ =$ceil(S_{page}/(IW * MSS))$. Assuming the cost is the TCP connection, then cost of each increased connection can be normalized to $C(n) = 1/ConnectionNumber_{\max}$, which indicates the cost caused by increasing connection number from $(n-1)$ to n connections. Similarly, the gain caused by increasing connection number from $(n-1)$ to n connections can be defined as $G(n)$ =(T_{HTTP}(n-1)-T_{HTTP}(n))/$T_{HTTP}(1)$. Then to determine the connection number, the client iteratively compares gain and cost per each increased connection until the cost is larger than gain.

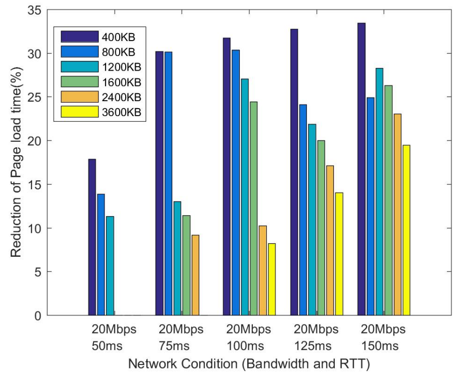

Fig. 5. Reduction of page load time with proposed algorithm

Figure 5 and Table I depicts the numerical results of page loading time reduction and corresponding number of connections by varying the size of page from 400KBytes to 3600KBytes and end-to-end RTT from 50ms to 150ms. In terms of the horizontal trend, as the network BDP increases, the page can experience more load time reduction and more connections can be opened. For example, when RTT is 50ms, the maximum improvement can be seen on 400KB page (only 17.9%) and pages no smaller than 1600KB experience no improvement. By contrast, when RTT increases to 150ms, the minimum time reduction across all pages is 19.5%. Similarly, regarding the impact of page size in the figure, more significant improvement can be observed for smaller pages. For example, smaller pages (less than 1200KBytes) can be optimized up to 33.5% across all network conditions. In contrast, time reduction of larger page is relatively smaller (from 9.2% to 26.3% when BDP reaches 20Mbps$*$150ms). Regarding the cost, which is number of parallel connections, more connections will be opened for smaller page or larger network BDP (ranging from 1 to 4). This range is still less than the typical parallel number adopted by HTTP 1.1/1.0 (6 parallel connections). These above observations indicate that our algorithm can be adaptive according the varying page load context and achieve a proper trade-off between time load reduction and the number of connections. We also observed that for the case that a client still selects one single connection (e.g. if page size is 3600KB, the available bandwidth is 20Mbps and the RTT is less than 100ms), the determination of establishing one single connection is based on the page load context, which means for specific page size and network condition, single connection is the better choice than any other number of connections.

IV. Prototype implementation and validation in emulated and real LTE network

In this section, we describe the prototype implementation of our framework and its performance in a local LTE network.

TABLE I. PAGE LOAD TIME REDUCTION

	20Mbps 50ms		20Mbps 75ms		20Mbps 100ms		20Mbps 125ms		20Mbps 150ms	
Page Size(KB)	*G(n)*	*n*	*G(n)*	*n*	*G(n)*	*n*	*G(n)*	*n*	*G(n)*	*n*
400	17.93%	2	30.22%	4	31.76%	4	32.76%	4	33.47%	4
800	13.94%	2	30.20%	4	30.42%	4	24.12%	3	24.95%	3
1200	11.40%	2	13.06%	2	27.12%	4	21.89%	3	28.31%	4
1600	0.00%	1	11.47%	2	24.46%	4	20.04%	3	26.30%	4
2400	0.00%	1	9.22%	2	10.25%	2	17.14%	3	24.60%	4
3600	0.00%	1	0.00%	1	9.48%	2	14.97%	3	19.49%	4

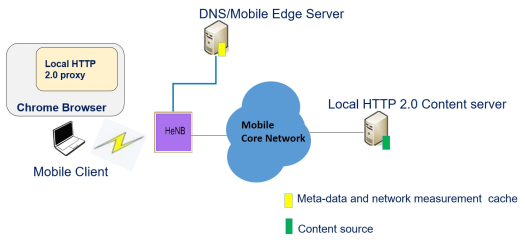

Fig. 6. Test Environment Setup

A. Prototype implementation and Environment setup

Figure 6 depicts the test-bed environment based on which the experiments were carried out. We modified the nghttp [30] proxy to enable adaptive connection numbers and embedded a meta-data query module which sends meta-data query to the MEC server along with the DNS query. This proxy is locally deployed at ubuntu 14.04.04 laptop to works as local proxy to the Google Chrome 47.0 Browser. Therefore all the HTTP/2.0 messages originated from the Chrome Browser will be intercepted by this local nghttp proxy. The nghttp content server is deployed behind the core network. The connection between server and client side proxy is configured as clear text HTTP 2.0, which is one of the two modes (h2 and h2c) in RFC 7540 [3]. Different from existing work like [4], [6], [16], our prototype only needs application and socket API level modification and still maintains the end-to-end service logic between mobile user and content provider, which leads to a low deployment cost and also being friendly to mobile operator. In our local LTE network, the average RTT is 65ms and average bandwidth is 32Mbps. We use tcpdump to trace the packet at client side in order to measure the page load time which is defined as the time from first handshake message sent to the arrive timestamp of the last data packet.

B. Performance test and Framework validation

We first evaluate our proposed scheme with random 15 pages selected from Top 200 websites in the Internet. Since we specifically focus on the performance in mobile network, we select its mobile version. The size of these pages ranges from 346KB to 3686KB and the number of objects ranges from 21 to 87. Considering the varying network conditions, we conduct back-to-back run of a page with and without our framework and each page was loaded 30 times. Figure 7 depicts the CDF plot of traced page loading time. It is apparent that our MEH framework significantly improves the overall page download time. In the median case, page loading time by standard HTTP/2.0 is 2.38s and it is reduced to 1.89s (reduced by 20.5%) with MEH. In addition, page loading time less than 2s occupies 38.4% without optimization and in contrast, 61.8% page load samples with optimization are finished within 2s. However, the improvement for large pages is not substantial. Similar percentages can be observed for the pages which need more than 3.5s to be loaded in both cases, which are 9.5% for standard HTTP/2.0 and 6.7% for optimized HTTP/2.0. This is because according to our algorithm, larger webpages experience low idle time during page load time therefore its improvement is much less than small pages. We also provide a detail comparison for 3 selected pages in box plot format in Fig. 8. Page 1 is 622KB with 32 objects, Page 2 is 1412KB with 72 objects and Page 3 is 3648Kb with 46 objects. From the plot box figure we can see that: (1) For small pages like Page 1 and Page 2, in the median case the performance gain are relatively higher (34.5% and 24.3%) while for larger page size (Page 3), the improvement is only marginal (6.03%). (2) we also observed that our algorithm leads to a small range of page load times for smaller pages (e.g pages 1 and page 2). This is because when the RTT increases, the page load time will increase. However, since the corresponding network BDP also rises, according to the analysis in section 3, our algorithm can help the client to select more connections to overcome the longer latency.

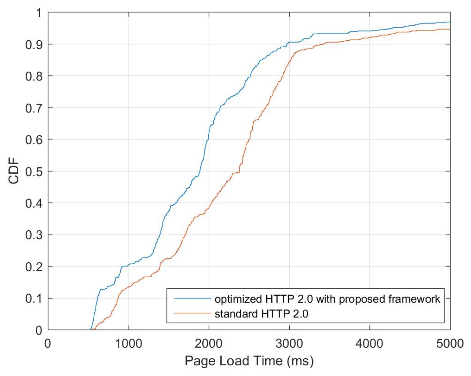

Fig. 7. Performance comparison of HTTP 2.0 without and with proposed framework in LTE network

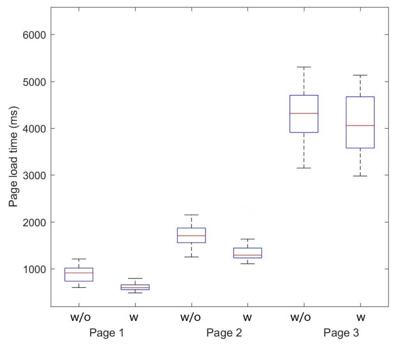

Fig. 8. Box plot of page load time without and with proposed framework

we also calculated the average size of mobile edge hint packets of top 200 mobile websites. Since we use gzip to compact the data and only selected fields are included in the packets, we observe that 89.8% of the edge hint are less than 1KB, which means compared to the total page size, the overhead and latency caused by these packets are minimal.

V. Conclusion

In this work, based on the idle time analysis of today's mobile webpage and mobile network condition, we present the framework of Mobile Edge Hint (MEH) to improve the page loading time. In this MEC-based framework, the MEC server which stands at the edge of network offline collects and maintains the context-aware metrics like external network latency and meta-data of popular web pages. Upon receiving requests on these popular web pages, the mobile edge is able to hint back to the HTTP/2.0 clients and the clients will execute a local algorithm to dynamically determine an optimal parallel connection number to download all objects embedded in the page, according to the hinted knowledge on page meta-data and the current network condition. These parallel connections can reduce the idle time during TCP slow start and the experiment results in a real LTE network show that the page load time can be improved up to 34.5% and in the median case, the improvement is 20.5%.

Acknowledgment

This work is funded by HEFCE Innovation Gateway project. The authors would also like to acknowledge the support of the University of Surrey's 5G Innovation Centre (5GIC) (http://www.surrey.ac.uk/5gic) members for this work.

References

[1] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDYier mobile web?" *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 2010–2023, 2015.

[2] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, "An in-depth study of LTE: effect of network protocol and application behavior on performance," in *ACM SIGCOMM Computer Communication Review*, 2013.

[3] M. Belshe, R. P. M. Thomson, and E. Mozilla, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, 2015.

[4] A. Sivakumar, S. Puzhavakath Narayanan, V. Gopalakrishnan, S. Lee, S. Rao, and S. Sen, "PARCEL: Proxy assisted browsing in cellular networks for energy and latency reduction," in *ACM on emerging Networking Experiments and Technologies*, 2014.

[5] S. Singh, H. V. Madhyastha, S. V. Krishnamurthy, and R. Govindan, "FlexiWeb: Network-aware compaction for accelerating mobile web transfers," in *MobiCom*, 2015.

[6] A. Sehati and M. Ghaderi, "Webpro: A proxy-based approach for low latency web browsing on mobile devices," in *IEEE IWQoS*, 2015.

[7] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?" in *USENIX Conference on Networked Systems Design and Implementation*, 2014.

[8] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. Choffnes, and R. Govindan, "Investigating transparent web proxies in cellular networks," in *Passive and Active Measurement*, 2015.

[9] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC 3390, 2002.

[10] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY really make the web faster?" in *IFIP Networking Conference*, 2014.

[11] J. Garcia, S. Alfredsson, and A. Brunstrom, "A measurement based study of TCP protocol efficiency in cellular networks," in *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2014.

[12] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with wprof," in *USENIX on Networked Systems Design and Implementation*, 2013.

[13] A. Sivakumar, V. Gopalakrishnan, S. Lee, S. Rao, S. Sen, and O. Spatscheck, "Cloud is not a silver bullet: A case study of cloud-based mobile browsing," in *Mobile Computing Systems and Applications*, 2014.

[14] N. Becker, A. Rizk, and M. Fidler, "A measurement study on the application-level performance of LTE," in *IFIP Networking Conference*, 2014.

[15] "Mobile edge computing a key technology towards 5G," White Paper, ETSI, 2015.

[16] F. Qian, V. Gopalakrishnan, E. Halepovic, S. Sen, and O. Spatscheck, "TM3: Flexible transport-layer multi-pipe multiplexing middlebox without head-of-line blocking," in *ACM CoNEXT*, 2015.

[17] M. Scharf, "Comparison of end-to-end and network-supported fast startup congestion control schemes," *Computer Networks*, vol. 55, no. 8, pp. 1921–1940, 2011.

[18] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin, "An argument for increasing TCP's initial congestion window." *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 3, pp. 26–33, 2010.

[19] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Characterizing web page complexity and its impact," *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, pp. 943–956, 2014.

[20] V. Gabale and D. Krishnaswamy, "Mobinsight: On improving the performance of mobile apps in cellular networks," in *International Conference on World Wide Web*, 2015.

[21] P. Blasco and D. Gunduz, "Learning-based optimization of cache content in a small cell base station," in *IEEE ICC*, 2014.

[22] chrome-har-capturer. [Online]. Available: https://github.com/cyrus-and/chrome-har-capturer

[23] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar, "KLOTSKI: Reprioritizing web content to improve user experience on mobile devices," in *USENIX Conference on Networked Systems Design and Implementation*, 2015.

[24] R. Netravali, A. Goyal, J. Mickens, and H. Balakrishnan, "Polaris: Faster Page Loads Using Fine-grained Dependency Tracking," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[25] F. Lu, H. Du, A. Jain, G. M. Voelker, A. C. Snoeren, and A. Terzis, "CQIC: Revisiting cross-layer congestion control for cellular networks," in *Mobile Computing Systems and Applications (HotMobile)*, 2015.

[26] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the performance of HTTP over several transport protocols," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 616–630, 1997.

[27] B. Han, S. Hao, and F. Qian, "Metapush: Cellular-friendly server push for http/2," in *Workshop on All Things Cellular: Operations, Applications and Challenges*, 2015.

[28] Y.-J. Lee and M. Atiquzzaman, "HTTP transfer latency over SCTP and TCP in slow start phase," in *IEEE ICC*, 2007.

[29] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic, "Parallel tcp sockets: Simple model, throughput and validation," in *IEEE INFOCOM*, 2006.

[30] NGHTTP. [Online]. Available: https://nghttp2.org/