Optimization of Webpage Downloading Performance with Content-aware Mobile Edge Computing

Peng Qian, Ning Wang, Bong-Hwan Oh, Chang Ge, Rahim Tafazolli
5GIC, Institute for Communication Systems, University of Surrey
Guildford, UK GU2 7XH
p.qian,n.wang,b.oh,c.ge,r.tafazolli@surrey.ac.uk

ABSTRACT

With increased complexity of webpages nowadays, computation latency incurred by webpage processing during downloading operations has become a newly identified factor that may substantially affect user experiences in a mobile network. In order to tackle this issue, we propose a simple but effective transport-layer optimization technique which requires necessary context information dissemination from the mobile edge computing (MEC) server to user devices where such an algorithm is actually executed. The key novelty in this case is the mobile edge's knowledge about webpage content characteristics which is able to increase downloading throughput for user QoE enhancement. Our experiment results based on a real LTE-A test-bed show that, when the proportion of computation latency varies between 20% and 50% (which is typical for today's webpages), the downloading throughput can be improved up to 34.5%, with reduced downloading time by up to 25.1%

CCS CONCEPTS

• **Networks → Middle boxes / network appliances**; **Network services**; *In-network processing*;

KEYWORDS

Mobile computing, webpage downloading, computation time

ACM Reference format:
Peng Qian, Ning Wang, Bong-Hwan Oh, Chang Ge, Rahim Tafazolli. Copyright 2017. Optimization of Webpage Downloading Performance with Content-aware Mobile Edge Computing. In *Proceedings of MECOMM'17, Los Angeles, CA, USA, August 21, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3098208.3098214

1 INTRODUCTION

Nowadays Hypertext Transfer Protocol (HTTP) based applications account for the majority of content traffic volume in mobile network environments [1]. The wide variety of HTTP applications such as web browsing, DASH (Dynamic Adaptive Streaming over HTTP) based video-on-demand and live-streaming, have distinctive features in terms of traffic patterns and user's Quality of Experience (QoE) models. Given the dynamicity and uncertainty of network resource availability in mobile environments, providing assured QoE for such types of application has become a distinct challenge. Among various strategies to tackle this challenge, making HTTP and its supporting transport layer protocols such as Transmission Control Protocol (TCP), Quick UDP Internet Connection (QUIC) [2] adaptive to dynamic network conditions and different content characteristics has become an attractive solution option.

In the literature, content and network adaptations on DASH-based video applications have been widely investigated. On the other hand, one unique feature of today's short-lived content applications (e.g. webpage downloading) is that computation time on the client device side accounts for non-negligible portion of the overall content access time given the significant growth of webpage complexity, especially due to the introduction of JavaScript files which requires a synchronous evaluation time [3, 4] in terms of parsing embedded objects within a webpage. Such a feature is expected to have substantial impact on the throughput performance of TCP which was designed to cater for network conditions rather than web content complexity involving large number of embedded objects with complex interdependency. With substantially growing complexity of web content nowadays, it will take a significant computation time for a mobile device to complete the parsing tasks of the embedded JavaScript files, and such computation latency typically blocks the network transmission and hence no follow-up objects can be downloaded until the parsing task is completed[3]. Recent research works have revealed that this computing latency can account for even more than 50% of the total content downloading time on mobile devices, due to the limited processing ability of mobile CPU, whereas its impact on more powerful terminals such as normal computers is much less of an issue [5]. As such, in a mobile network environment, this computation latency will substantially degrade the downloading throughput as well as the user's QoE in terms of content access delay [4, 5].

In order to eliminate, or at least to minimize this negative effect introduced from computation latency, the main research question to be tackled in this paper is *how to compensate the throughput gap caused by the computation latency during short-lived application loading by only adapting the transport-layer protocol?* The benefit of addressing this issue at transport layer is that it is not affected by any application layer constraints such as HTTPs content encryption and security policy. On the other hand, it has been recently advocated to embed network intelligence at mobile edge in order to enable end-to-end performance enhancements at the application layer, and the ETSI Mobile Edge Computing (MEC) paradigm [6] is one of the representative examples. The application of MEC can be either direct involvement of user/data plane operations such as in-networking content caching and prefetching [7], or alternatively without breaking the end-to-end connection but instead only by

MECOMM'17, Los Angeles, CA, USA
 ISBN 978-1-4503-5052-5/17/08…$15.00
DOI: http://dx.doi.org/10.1145/3098208.3098214

providing necessary knowledge (typically network conditions) back to the application layer which is able to perform adaptive operations against various dynamicity [8]. In order to address the aforementioned technical problems, this paper introduces a MEC-based solution with extended knowledge for enhancing short-lived content application performances, including not only network conditions but also context-information about (popular) web content characteristics, more specifically the required computation time/complexity on the device side for processing the webpage during downloading operations. As such, both types of complementary context information can be fed back to application clients (e.g. a mobile device) in order to execute smart operations at the transport layer, specifically through the optimization of TCP initial window (IW) size. This is particularly useful giving that for short-lived applications such as webpage downloading, where content downloading is normally completed during the TCP slow-start phase. On the other hand, compared to the MEC-based reporting of network conditions, the mobile edge's awareness of the computation time on the device side has not been considered, but it does play an important role in webpage downloading performance. In this context, we propose a MEC-based solution with a plug-and-play optimization algorithm based on harvested knowledge on both network conditions and device computation time for given (popular) webpages in the Internet. To our best knowledge, this is the first piece of work that proposes content-aware mobile edge computing in order to tackle the performance deterioration of web content applications due to the introduction of webpage computation latency.

We have implemented the proposed MEC-based solution and evaluated it in our real LTE-A testing infrastructure. Since such a technique can be applied in both traditional TCP and the emerging QUIC scheme that share the same congestion control mechanism [2], we use QUIC as the underlying protocol as it offers a better implementation flexibility and also increasing popularity today. Our experiment results reveal that, when the computation latency accounts for less than 20% of the overall downloading time, the throughput gap can be fully compensated. When the proportion of computation latency varies between 20% and 50%, the throughput can be improved up to 34.5%. Such improved downloading throughput has led to the reduced webpage downloading time by up to 25.1%.

2 MEC FRAMEWORK OVERVIEW FOR WEBPAGE DOWNLOADING ACCELERATION

As previously mentioned, the design goal of the MEC-based solution is to achieve optimized webpage content downloading performance by intelligently setting TCP/QUIC IW sizes according to the device's knowledge on network conditions and webpage processing complexity. It is worth noting that such embedded computing tasks often have dependency between individual content object downloading, meaning that some webpage objects cannot be downloaded until the webpage processing task has been completed [3, 4]. In addition, due to the relatively low CPU resources capabilities on mobile devices, the impact on TCP/QUIC performance from the time spent on the webpage processing cannot be neglected. In Section 3 we will provide detailed elaboration on: 1) how such computation

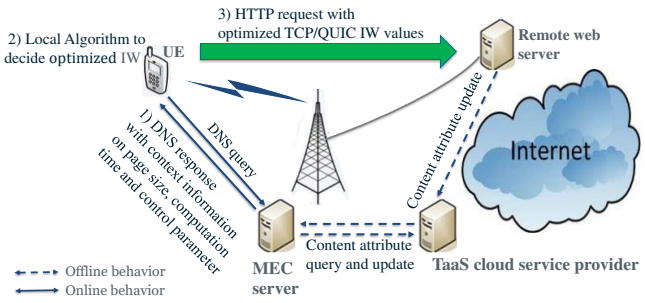

Figure 1: Overview of proposed MEC framework

latency may affect the content transmission throughput and downloading performance, and 2) our proposed scheme to mitigate such a problem with a TCP/QUIC-based optimization technique that is executed at the mobile device side. Before introducing the theory part of the solution, in this section we first describe how a mobile device can obtain necessary knowledge as input for enabling such an optimization through the help from a MEC framework.

Specifically required attributes include the information of total webpage size and the computation latency to be experienced by the mobile device which represents the network idle time while delivering webpage objects. It is worth mentioning such information is difficult to be obtained directly from mobile clients which previously visited that webpage, since 1) a mobile divide cannot explicitly report out this computation latency and 2) the webpage attribute can be updated since previous visit by other clients. In addition to the knowledge about content attributes, the MEC server also needs to achieve an optimized trade-off between individual QoE performances and the overall network resource utilization. Towards this end, the MEC server also passes on to the mobile a specific control parameter as input to calculate the optimized IW size, but without incurring resource competition between concurrent downloading sessions. Detailed description on such a parameter will be introduced in Section 3.

Figure 1 illustrates the proposed framework in which a MEC server is responsible for obtaining necessary context information and feeding parameters to the mobile devices for optimizing webpage downloading performances. We first focus on how the MEC server is able to acquire its knowledge about webpage content attributes. First of all, it is worth mentioning that practically it is not possible or necessary to have MEC involved in every single webpage in the public Internet due to scalability concerns. Instead, such an approach is mainly useful for accessing popular webpages with a large number of embedded content objects, (e.g. the default webpages of news websites). According to [7, 9, 10], the concept of "Testing as a service (TaaS)" has been promoted by cloud service providers in the sense that they are able to perform specific testing/measurement services upon requests from any stakeholders in the public Internet. For instance, if a MEC server determines to perform downloading acceleration of a webpage which is frequently visited by its local clients, it may decide to obtain the information on the attributes of the webpage, including the computation latency by typical mobile devices and the webpage size. Towards this end, the MEC server can issue a customized script to a TaaS cloud test platform (e.g. Google firebase [11]) or a trusted remote proxy (e.g.

Google Flywheel [10]). Furthermore, deep web content loading analysis tool like WProf-M [5] can be easily integrated to such platforms in order to return the detailed webpage loading procedure. By leveraging such TaaS cloud content analysis platforms, the computation latency can be obtained and maintained at the MEC server side. Also it is important to note that, according to measurements conducted in [5], for a given webpage the actual computation latency is approximately the same for smart mobile phones with different CPU computing capabilities, while such latency is very much different for computers/laptops. This observation suggests that one single obtained computation latency value for a given page can be applicable to a wide range of mobile phones. On the other hand, such an approach is mainly used for low/medium CPU load scenarios, since high CPU load on the mobile phone side will get the computation time become the major bottleneck so that even increasing the TCP/QUIC IW size on the network side will not help much. Another content attribute that needs to be obtained by the MEC server is the total size of the webpage, and in fact such information is also required by the TaaS cloud service provider responsible for measuring the computation latency by mobile phones. In this case, a web-based content analysis tool needs to be deployed at the web server side, and upon a significant change in the total size of a webpage under consideration, the web server will notify the cloud service provider about such update through mechanisms such as that is provided in [12].

According to [6], a MEC server can be embedded with a DNS function such that a URL resolution request from a mobile phone can be handled locally at the MEC server. According to our proposed scheme, apart from the conventional DNS mapping the MEC server also maintains content attributes of popular requested URLs, including the up-to-date total webpage size and the computation latency, both of which are updated from the TaaS cloud server. Once receiving a DNS query from a mobile client, the MEC server will resolve the request and also return the recorded content size, estimated computation latency, together with a policy control parameter α (explained in Section 3) through the DNS response message. Upon receiving the DNS response, the client will perform a handshake to the resolved web server and hence calculate the round trip time (RTT) by inspecting the corresponding handshake messages. Such information obtained directly from the mobile device will also be used as input for determining the optimized IW value. With all necessary input, an optimal IW can be computed by our proposed algorithm that is executed at the mobile phone side (see details in Section 3), and such a value will be embedded in the very first HTTP message to be enforced by the remote web server.

3 TACKLING PERFORMANCE DETERIORATION CAUSED BY COMPUTATION LATENCY

From the perspective of the network side, the major negative effect caused by the computation latency is that it unnecessarily prevents the congestion window from increasing during the slow start phase, and hence limits the data transmission throughput even when sufficient bandwidth is available. To tackle such a problem, we propose an efficient algorithm to determine an optimal IW size which takes into account the trade-off between the increased initial bandwidth consumption and the QoE improvement (defined as user-perceived content completion time [13]).

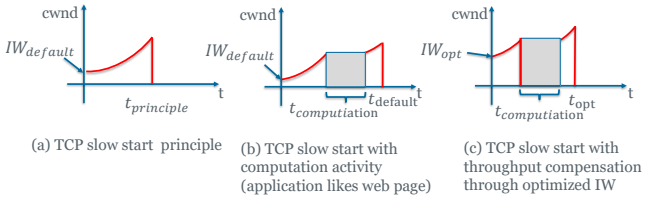

Figure 2: Throughput gap due to computation activity

We use a set of simple diagrams in Figure 2 to explain how the computation latency affects the overall throughput performance and how such inefficiency can be potentially compensated by optimizing IW size. Figure 2(a) depicts the normal exponential congestion growth during the slow start phase, but without any necessary computation time involved. The throughput in this simple case can be represented as $th_{principle} = \frac{S}{t_{principle}}$, where S is the content size and $t_{principle}$ is the time for transmitting this content. However, for a short-lived application (e.g. web-based application), a computation activity may exist in the middle of the slow start phase [3]. As it is shown in Figure 2(b), this computation activity blocks the data transmission at the network side. In this case, according to the default TCP behaviour, the actual completion time of a webpage $t_{default}$ includes both the data transmission time and computation time, which results in sub-optimal overall throughput performance $th_{default} = \frac{S}{t_{default}}$ as compared to the principle throughput in Figure 2(a). In order to maximally compensate this throughput gap ($th_{principle} - th_{default}$), our main strategy is to accelerate the data transmission of this short-lived application with embedded computation time through an increased initial congestion window size. It can be seen in Figure 2(c) that for a given content size S, if the completion time of this webpage is shorten to t_{opt} which is expected to be as close as $t_{principle}$, then the throughput gap can be minimized, and ideally if $t_{opt} = t_{principle}$ (meaning that the total computation time can be fully compensated), then the optimized throughput $th_{opt} = \frac{S}{t_{opt}}$ can be retained same as the normal TCP situation. Towards this end, the key technical challenge is the knowledge on the actual computation latency in order to determine the IW size. According to our proposed framework, such knowledge is obtained from the MEC-based technique we introduced in Section 2, and now we formally present the algorithm for determining the optimized IW size baesd on such knowledge.

3.1 Context-aware throughput compensation algorithm

The proposed throughput compensation algorithm (see in Algorithm 1) includes two steps: the first step is find an adequately large IW size which can minimize the throughput gap and this increased IW size will be set as IW_{max} which is the upper bound of the final optimal IW size. The second step is to search an optimal IW_{opt} which takes into account the trade-off between QoE improvement and increased initial bandwidth consumption.

In the first step, at client side, once receiving the knowledge of content size, the computation latency from the MEC server and

the measured RTT from handshake messages, the principle transmission time $t_{principle}$ will be estimated. Given that webpages typically have a small content size, it can be assumed that the corresponding transmission normally finishes within the slow-start phase. Therefore we borrow the model [14] to approximately estimate the $t_{principle}$ as:

$$t_{principle} = RTT * (\lceil \log_{\gamma} \left(\frac{S}{\mathrm{IW}_{\mathrm{default}} * s_{mss}} \right) \rceil), \quad (1)$$

where RTT is the Round Trip Time between the client and the web application server, the $\mathrm{IW}_{\mathrm{default}}$ is the initial congestion window with the default value of 10 according to [15]. s_{mss} is the TCP maximum segment size and γ is a constant 1.5 if the delayed ACK is enabled.

The estimated $t_{principle}$ for a content can be obtained by Eq. (1) where the IW size is set to $\mathrm{IW}_{\mathrm{default}}$. If $t_{principle} \leqslant t_{computation}$, it indicates that retaining the principle throughput is impossible, and in this case, in order to minimize the throughput gap, all content data should be sent in the first round, therefore the IW size should be set as $\mathrm{IW}_{\mathrm{max}} = S/s_{mss}$. If $t_{principle} > t_{computation}$, given that the content size, computation latency from MEC server and inspected RTT from handshake messages are the same for the same content, in order to achieve same throughput performance as the normal TCP without any disruption from computation latency, a larger IW size rather than $\mathrm{IW}_{\mathrm{default}}$ can be computed to reduce the data transmission time towards the target of $(t_{principle} - t_{computing})$ according to Eq. (1). We define the function which obtains the IW as $f(t_{computation}, S, RTT)$ for the calculation of this larger IW size.

However, at the application layer, the eventual benefit of this throughput compensation results in a better user QoE but it comes at a cost of a larger initial bandwidth consumption. In order to retain the benefit of compensating the throughput as well as that of minimizing the impact of initial bandwidth consumption, in the second step, we use a threshold parameter α as a configurable threshold to balance the trade-off between the improvement of user QoE and increased initial bandwidth consumption, and such a value is typically returned from the MEC server according to specific network policies. The rationale behind this control parameter is that from Eq (1) and [13], the QoE improvement requires an exponentially increased IW size, hence it is not desirable to set the IW size to an excessively large value. We borrow the QoE model from [13] to define the QoE improvement by an increased IW size as

$$\Delta QoE(\mathrm{IW}_{\mathrm{opt}}, S, t_{computation}, RTT) = \ln \left(\frac{t_{default}(\mathrm{IW}_{\mathrm{default}}, S, t_{computation}, RTT))}{t_{opt}(\mathrm{IW}_{\mathrm{opt}}, S, t_{computation}, RTT))} \right) \quad (2)$$

And the increased initial bandwidth consumption can be defined as

$$\Delta Cost(\mathrm{IW}_{\mathrm{opt}}, RTT) = (\mathrm{IW}_{\mathrm{opt}} - \mathrm{IW}_{\mathrm{default}}) * S_{mss}/RTT.$$

Then the tradeoff function between QoE improvement over the increased initial bandwidth consumption can be expressed as:

$$\Delta QoE(\mathrm{IW}_{\mathrm{opt}}, S, t_{computation}, RTT)/\Delta Cost(\mathrm{IW}_{\mathrm{opt}}, RTT).$$

According to the algorithm, to find a optimal $\mathrm{IW}_{\mathrm{opt}}$, a temporary variable $\mathrm{IW}_{\mathrm{test}}$ is set to $\mathrm{IW}_{\mathrm{default}}$ as a starting point and this $\mathrm{IW}_{\mathrm{test}}$ gradually increases by multiplying γ within the range from $\mathrm{IW}_{\mathrm{default}}$ to $\mathrm{IW}_{\mathrm{max}}$ in each iteration. Once the corresponding $\frac{\Delta QoE}{\Delta Cost}$ is less than the pre-configured threshold parameter α, the algorithm for compensation will stop and we set the final $\mathrm{IW}_{\mathrm{opt}}$ accordingly (see in Algorithm 1).

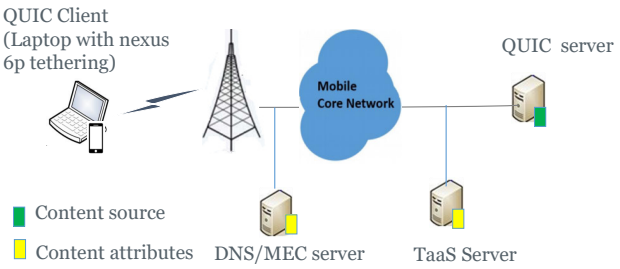

Figure 3: Proof-of-concept implementation in LTE-A test bed

Algorithm 1 Algorithm for Throughput compensation

if $t_{\mathrm{computation}} >= t_{\mathrm{principle}}$ **then**
  $IW_{\mathrm{max}} = S/s_{\mathrm{mss}}$
else
  $IW_{\mathrm{max}} = f(S, t_{\mathrm{computation}}, RTT)$
end if

$IW_{\mathrm{test}} = IW_{\mathrm{default}}$, i=0;
while $IW_{\mathrm{test}} < IW_{\mathrm{max}}$ **do**
  $IW_{\mathrm{test}} = min(IW_{\mathrm{test}} * \gamma, IW_{\mathrm{max}})$;
  if $\Delta QoE/\Delta Cost < \alpha$ **then**
    break;
  end if
  i++;
end while
$IW_{\mathrm{opt}} = min(IW_{\mathrm{default}} * \gamma^{i}, IW_{\mathrm{max}})$

4 IMPLEMENTATION AND PERFORMANCE IN REAL LTE/LTE-A NETWORK

4.1 Proof-of-concept implementation

In order to realistically evaluate the proposed MEC framework and the optimization algorithm, we implemented and tested the solution in a real LTE-A based test-bed infrastructure, as shown in Figure 3.

At the client side, following the same access setup manner in [5], we use a laptop (Ubuntu 14.04) tethering a nexus 6p phone attached to the locally deployed LTE-A network. We fork the QUIC client code from Chromium 52 and embed our IW optimization algorithm as well as the extended DNS query interface to the client code. A MEC server with local DNS cache is deployed just between the base station and the mobile core network. The content is located at a remote HTTP 2.0/QUIC server. The webpage content under testing contains controllable computation latency (from 5% to 95% of the total expected content completion time) embedded in the sequences of content objects requests. In order to provide the knowledge on content attributes to the MEC server, a TaaS server is deployed behind the core network and it periodically loads the content at the remote server and outputs the HTTP 2.0 headers and the computation latency with an embedded script. At the MEC server, the control parameter α is set to 1/3 and we leave the adaptation of control parameter as our future work. With respect to the network

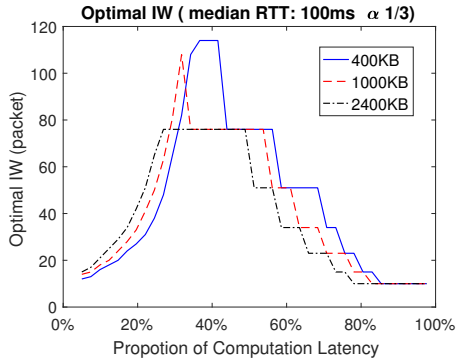

Figure 4: Optimal IW in real LTE-A network (varying RTT)

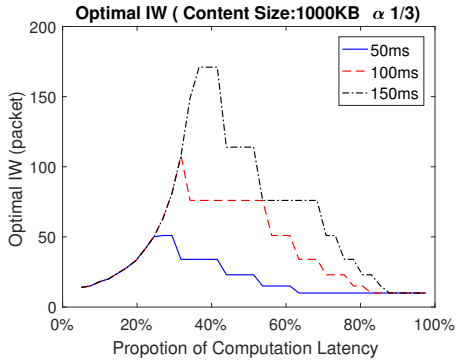

Figure 5: Optimal IW in real LTE-A network (varying content size)

conditions, the average RTT at the LTE-A radio interface is 25.2ms and the average bandwidth is 60.2Mbps. We also set the content size to 400KB, 1000KB, 2400KB and vary the latency from the base station to the content server as 25ms, 75ms and 125ms in order to emulate different latency in a real public Internet environment. All the experiments are executed back-by-back for 30 times and the median value of throughput as well as the completion time are printed by an embedded print script in QUIC client code.

To evaluate the impact of computation latency on the optimal IW size according to our proposed algorithm, in the following figures in this section, we set x-axis to the proportion of computation latency which is obtained by a combination of estimated data transfer time (from Eq. (1)) and the computation latency received from MEC server. Figure 4 shows the relationship between the proportion of computation latency and optimal IW size in a real LTE-A network with fixed RTT of 100 ms. It can be observed that the optimal IW size reaches its highest values when the computation accounts for 30%-40% of overall latency. For the content size of 400KB, the optimal IW size reaches its peak value 114 when the proportion of computation latency is 35%. When the proportion of computation latency rises to 85%, the optimal IW reduces to 10 which is the default value. For a medium (1000KB) and larger (2400KB) content size, their peak congestion window reduce to 108 and 76, respectively. Figure 5 shows the relationship between the proportion of computation latency and optimal IW size with different RTTs. It can be seen that all the three curves behave in similar patterns. Since the control parameter α targets to align the cost of the starting bandwidth consumption, a larger RTT (150ms) will allow the client to select a larger peak (171) congestion window and vice versa.

4.2 Throughput compensation and improvement on downloading time

Figure 6 and 7 show the median throughput compensation by applying our algorithm with different RTT values and content sizes, respectively. It can be observed in both figures that when the computation latency is less than 20%, the throughput gap can be approximately fully compensated and we call this period as "full compensation scenario". When the proportion of computation latency increases from 20% to 50%, the compensated throughput increases from 22.4% to 33.5% and then continuously decreases to 18.2%. We call this interval as "partial compensation scenario". Then the compensated throughput drops below 15% when the proportion of computation latency increases above 50% and this interval

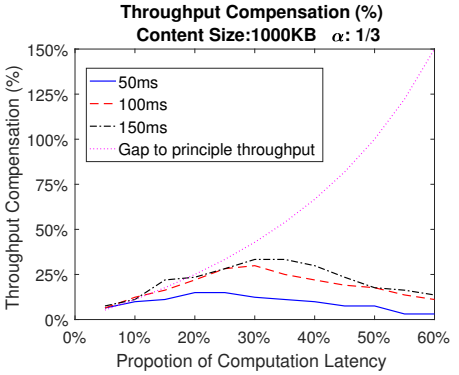

Figure 6: Throughput compensation in LTE-A network (varying RTT)

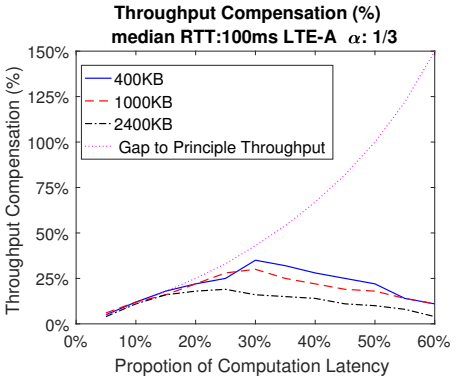

Figure 7: Throughput compensation in LTE-A network (varying content size)

can be called "minor compensation scenario". Specifically, figure 6 shows that a larger RTT has a larger maximum throughput compensation (33.2%) which is closer to the throughput of the principle TCP. Figure 7 shows that, compared to the larger content sizes, a smaller content (e.g. 400KB) can reach a higher peak throughput compensation (34.5%) and in contrast, larger content (2400KB) has a comparatively smaller maximum throughput compensation (19.6%).

With regard to the throughput compensation algorithm, we apply the analytical model [14] to benchmark the data transfer time and corresponding proportion of computation latency. Figure 8 and 9 show that, in general, the difference between the estimated and

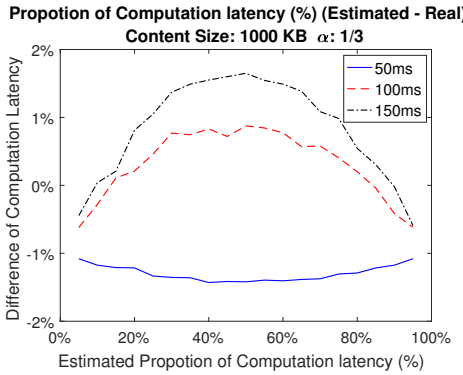

Figure 8: Difference between real and estimated proportion of computation latency (varing RTT)

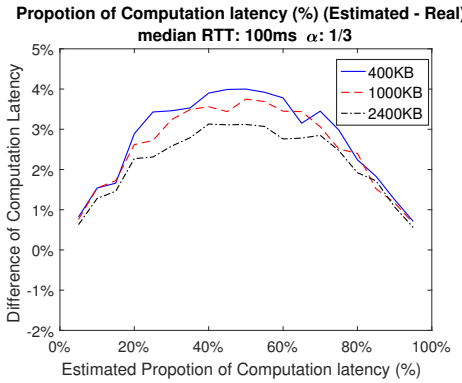

Figure 9: Difference between real and estimated proportion of computation latency (varing content size)

real proportion of computation latency ranges between -0.84% and 5.30%. When the estimated proportion of computation latency is around 50%, the difference between estimated and real proportion of computation latency reaches its maximum point. We attribute this minor gap to the nature of the fluctuated network condition [1]. Consequently, an under-estimated/over-estimated proportion of computation latency will lead to smaller/larger IW size selection, thus the real throughput compensation is slightly less/larger than the expected value accordingly. Based on our analysis on dataset, the difference between the estimated throughput compensation and the real throughput compensation fluctuates between -6.3% to 2.8%.

4.3 User-perceived application completion time

From the perspective of the application layer, the ultimate target by improving the principle throughput at transport layer is to improve the actual webpage downloading time. According to our testing results, for the "fully compensation scenario", the web-based application can experience an equal completion time as with no computation latency involved. For the "partial compensation scenario" scenario, the completion time can be improved by up to 25.1% and for the "minor compensation scenario" scenario, this completion time reduction becomes less significant. If combining the improvement from application layer approaches like network edge pushing and caching, note that the upper bound throughput of these approaches is same as the principle throughput at transport layer, but in practice, these approaches will be limited by the content structure and security concern. Additionally, in this case, there will be an overlapping between network activity and computation activity which is out of scope, thus we leave the investigation of this topic to our future work.

5 CONCLUSION

In this paper we proposed a MEC framework for guiding mobile clients to optimize web content downloading performance with the consideration of computation latency. In this framework, a MEC server collects and distributes necessary content attributes together with a necessary threshold parameter in order to direct the client to execute a novel algorithm for determining an optimal IW size. To the best of our knowledge, this is the first piece of work that addresses how a content-aware MEC-based approach can be applied for optimizing web content downloading performances. The experimental results from a real LTE-A testbed reveals that if the computation latency accounts for a small proportion (< 20%), the throughput gap can be fully compensated, and if the proportion of computation latency stays between 20% to 50%, the throughput gap can be partially compensated by up to 34.5%.

ACKNOWLEDGMENTS

This work is partially funded by the EPSRC CONCERT Project (EP/L018683/1). 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] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *10th international conference on Mobile systems, applications, and services*, pages 225–238. ACM, 2012.
[2] QUIC, https://www.chromium.org/quic.
[3] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. How Speedy is SPDY? In *USENIX Conference on NSDI*, 2014.
[4] Jamshed Vesuna, Colin Scott, Michael Buettner, Michael Piatek, Arvind Krishnamurthy, and Scott Shenker. Caching Doesnt Improve Mobile Web Performance (Much). In *USENIX ATC*. USENIX Association, 2016.
[5] Javad Nejati and Aruna Balasubramanian. An in-depth study of Mobile Browser Performance. In *25th International Conference on World Wide Web*, 2016.
[6] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing – A key technology towards 5G. *ETSI White Paper*, 11, 2015.
[7] Ashiwan Sivakumar, Shankaranarayanan Puzhavakath Narayanan, Vijay Gopalakrishnan, Seungjoon Lee, Sanjay Rao, and Subhabrata Sen. PARCEL: Proxy assisted browsing in cellular networks for energy and latency reduction. In *ACM on emerging Networking Experiments and Technologies*, 2014.
[8] P Qian, N Wang, G Foster, and R Tafazolli. Enabling context-aware http with mobile edge hint. In *IEEE CCNC*, 2017.
[9] Ali Sehati and Majid Ghaderi. WebPro: A proxy-based approach for low latency web browsing on mobile devices. In *IEEE 23rd IWQoS*, pages 319–328. IEEE, 2015.
[10] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. Flywheel: Google's Data Compression Proxy for the Mobile Web. volume 15, pages 367–380, 2015.
[11] Google Firebase, https://firebase.google.com/.
[12] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *13th USENIX Symposium on NSDI*. USENIX Association, 2016.
[13] Sebastian Egger, Peter Reichl, Tobias Hoßfeld, and Raimund Schatz. "Time is bandwidth?" Narrowing the gap between subjective time perception and Quality of Experience. In *IEEE ICC*, pages 1325–1330. IEEE, 2012.
[14] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling tcp latency. In *IEEE INFOCOM*, volume 3, pages 1742–1751. IEEE, 2000.
[15] IETF 6928:Increasing TCP's Initial Window.