How Edge Computing and Initial Congestion Window Affect Latency of Web-based Services: Early Experiences with Baidu?

Qingyang Zhang*, Hong Zhong*, Jiaoren Wu† and Weisong Shi‡
*School of Computer Science and Technology, Anhui University, Hefei, China, 230601
†Baidu Online Network Technology (Beijing) Co., Ltd., Beijing, China, 100080
‡Department of Computer Science, Wayne State University, Detroit, MI, USA, 48202
qyzhang@wayne.edu, zhongh@ahu.edu.cn, wujiaoren@baidu.com, weisong@wayne.edu

***Abstract*—More and more things, generating huge data, will come into and enrich our lives, and the Web of Things (WoT) as a guide allows these things to be part of the World Wide Web (WWW), by using various data analysis services on the WWW. However, based on our observation on the image recognition and searching service of Baidu, pure image data transmission costs hundreds of milliseconds, besides the time of connection establishment. Inspired by the emerging Edge Computing, we analyzed the relationship between time consumption and different service provider's locations, as well as different initial congestion windows of the Transmission Control Protocol (TCP), which affect web-based services' performance. Based on our experiments in different scenarios (*i.e.,* initial congestion window, speed of connection device and server location), we found that pushing services to the edge of network and increasing initial congestion window, both of them can reduce latency on connection establishment and data transmission, especially when users are traveling at a high speed.**

***Index Terms*—Edge computing; Web service; TCP; Congestion control; Initial congestion window size.**

1. Introduction

As the rapid growth of the Internet of Things, billions of geographically distributed things are connected to our Internet. By 2020, estimated by Cisco [1], 50 billion things will connect to the network and generate 507.5 zettabytes (ZB) data per year and by 2021, the data size will sharply increase to 847 ZB per year. The WoT [2] is used to allow such amount of things to be connected and update generated data to various data analysis services hosted on the WWW. For example, the speed recognition service, hosted on the WWW, serves as the back-end service of Alexa, and thus people can talk with Alexa via Amazon Echo [3] to control other smart things in the home. However, various data analysis services, especially recognition services (*e.g.,* voice recognition services and image recognition services), require raw data to be uploaded to a remote cloud via web protocols (*i.e.,* HyperText Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS) and other TCP-based protocols, such as Message Queuing Telemetry Transport (MQTT)), which has a powerful computing resource and artificial intelligence models with powerful capacity on recognition. Thus, volume data are transmitted, resulting in a high response latency, which is too large to affect user experiment. For example, it is oppressive that chatting with smart things, *i.e.,* Amazon Echo or Google Home, need to wait for several seconds.

The emerging Edge Computing (a.k.a, fog computing [4], cloudlet [5]), referring to "the enabling technologies allowing computation to be performed at the edge of the network" [6], is promising to reduce end-to-end latency. In this case, pushing such recognition services in the WWW to the proximity of data sources results in lowing response latency. Many applications [7], [8], [9] have benefited from edge computing. For example, Drolia *et al.* [8], [10] proposed an edge-caching for image recognition services with a lower latency than cloud-based solutions, which enables image recognition on edges and reduces computing time and response latency. Based on our observation on an image recognition and search service of Baidu, one of the large-scale search engines in the world, which requires user uploading their source image thus response some related information (*e.g.,* similar images and recognition results), and costs hundreds of milliseconds via a 4G cellular network, with a poor user experience, there are two main factors affecting response latency for these web-based services, including multiple-round handshakes, round-trip time (RTT) and congestion window size of the TCP protocol, which is underlying protocol of web protocols. In this case, we first measured the effect on reducing response latency, including handshaking latency and data transmission latency, by leveraging edge computing-based techniques.

As another factor affecting the web performance [11], [12], the congestion window determines the number of bytes that can be outstanding at any time, thus a small initial congestion window (`init_cwnd`) requests multiple rounds of data transmission when uploading volume data, *e.g.,* short video, resulting in a high data transmission latency. Google has proposed to increase this value from 3 to at least 10 in 2010 [13]. However, as the growth of transmitted data size and network reliability, the increased value of 10 now is small for most of the applications, and it should be

978-1-5386-9445-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SEC.2018.00052

considered to increase again to meet the challenge of high data transmission latency or enable applications to adjust it dynamically. Therefore, we also measured the performance via the cellular network with different initial congestion windows.

The rest of the paper is organized as follows. We describe the problem statement that includes the latency analysis of an image recognition service traces in Section 2. We first introduce the procedure of that image service and then analyze the relationship between RTT, the initial congestion window of the TCP and response latency in Section 3. Section 4 focuses on experiments and results. Finally, we conclude in Section 5.

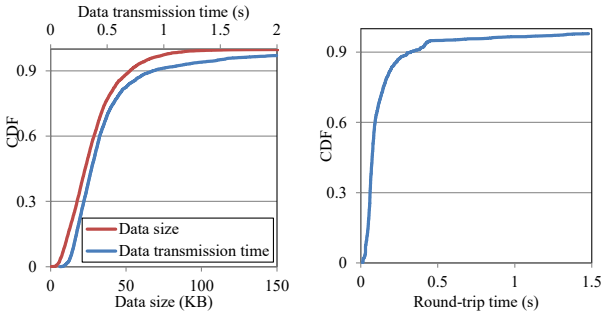
2. Problem Statement

As the increasing number of smart things connected to the Internet, various multimedia data, such as text, voice, image, short video and living video, are generated and transmitted to the cloud for recognition, such as speech recognition, image recognition, and video analysis. For example, in the smart city case, IP cameras upload their living videos or detected person areas to a private cloud for face recognition. In this case, the Table 1 shows sizes of text, voice, image and short video in Wechat [14], a popular instant chatting application in China, as well as the sizes of living videos with different resolutions based on our previous work [15], [16]. Note that the resolutions of the compressed image and video are low-quality, in terms of resolution (*i.e.,* 960×540) and compression ratio.

TABLE 1. SIZES OF TEXT, VOICE, IMAGE AND SHORT VIDEO.

Type	Description	Size
Voice	Adaptive multi-rate encoded	0.9-1.2KB/s
Image	Compressed high-defined image	50-200KB
Image	HD image with low compression	2-4MB
Short video	H.264-encoded	100-160KB/s
Living video	720P and H.264-encoded	3.8Mbps
Living video	1080P and H.264-encoded	5.8Mbps
Living video	2160P and H.264-encoded	19Mbps

However, in WoT era, to analyze multimedia data leveraging these data analysis services on the cloud (either a private cloud or a public cloud), things must connect to the cloud via different network access modes, such as Wide Area Network (WAN), Local Area Network (LAN), WiFi, Bluetooth or cellular network, and then upload their data to cloud-based services for recognition via web protocols, which results in a high response latency. Our observation on an HTTPS-based similar image searching service of Baidu [17], which can recognize and find some similar images in the database for the user, shows that the data transmission, including data uploading and data downloading, costs hundreds of millisecond using a 4G cellular network, in average. Note that, it does not include the time on TCP handshaking and Transport Layer Security (TLS) handshaking of the HTTPS connection, which also costs hundreds of milliseconds, as well as the processing time on the server, which costs around one and half seconds. Such a high response latency leads to a terrible user experience, which might reduce the usage rate of this service. The Fig. 1(a) illustrates the cumulative distribution function (CDF) of data transmission latency, as well as the size of these data, based on our analysis of Baidu's traces. Note that we did not figure out the time according to the image size, and the average data transmission latency is around 550 ms when the size of most of the images is lower than 150KB. The time is mainly distributed between 200 ms and 800 ms, and the higher data transmission latency in the Fig. 1(a) is caused by a weak network, in which the connection has a high packet loss rate, low bandwidth, and high connection latency. To measure the network quality, we captured part of the service traffics in server side using `tcpdump` and then analyzed captured data. The analyzed result of round-trip time is shown in the Fig. 1(b). It can be seen that there are some connections having a high round-trip time (up to 0.5 seconds), which results in a high data transmission latency since multiple-RTT data transmission is required. The detailed analysis will be introduced in Section 3.



(a) Latency and data size. (b) RTT.

Figure 1. Cumulative distribution function.

3. Problem Analysis

In this section, we analyzed the procedure of the mentioned service to know how edge computing and the initial congestion window can affect data transmission latency.

The HTTPS is the secure version of HTTP encrypted by TLS, thus an HTTPS connection establishment includes TCP handshaking, TLS handshaking. The Fig. 2 shows the procedure of Baidu's image service, including the TCP handshaking, TLS handshaking and data transmission via a slow-start TCP. After the TCP establishment between the similar image searching service and the user device, it starts to TLS handshaking for HTTPS connection. Then, the mobile device downloads a configure file for the service. Normally, taking a picture will occupy at least one second, and thus the congestion window size has reduced to the initial value (3 in our figure, but 10 in the real service). Note that, if there is no interval time between connection establishment and data uploading, the congestion window

size is still initial value, since the TCP and TLS handshaking usually cannot make congestion window increment. Once the image is prepared, it will be uploaded to the service and waiting for long processing time in the server, the result, hundreds of bytes data including a URL, will be backed to the user device. Here, the data transmission latency is the summation of data uploading latency and result downloading latency, where the former will cost several and half RTTs, and later costs half RTT.

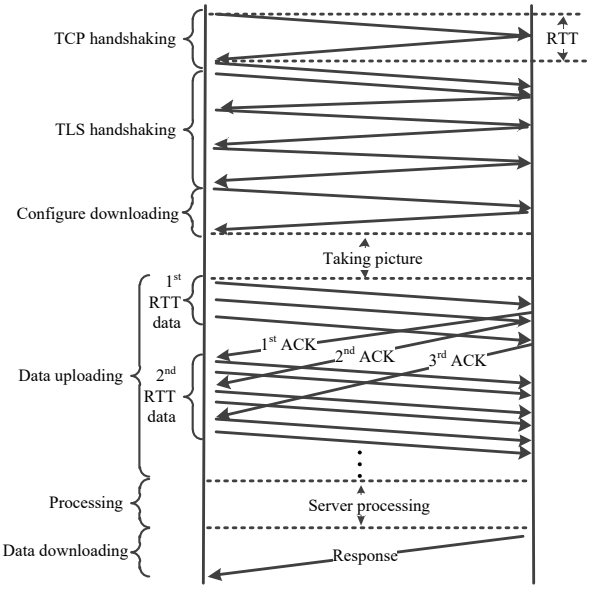


Figure 2. The procedure of the image service.

First of all, the TCP handshaking and TLS handshaking usually cost one and a half RTT and three RTTs, respectively, resulting in hundreds of milliseconds latency[1]. Thus, the RTT is a key factor for time consumption on HTTPS connection establishment. Besides, it is a key factor as well for data transmission, which we analyzed in detail later. In the data transmission phase, as shown in the Fig. 2, the initial congestion window determines the number of bytes for a sender at the first round, when not receiving any acknowledgment. And after one RTT, the sender will receive the acknowledgments (ACKs) for initial three packets and increase congestion window to 6. The value of the congestion window will be increased by one with each ACK received, effectively doubling the window size each RTT. Thus, with the `init_cwnd`of 3, the sender can send three data packets all with the maximum segment to the receiver, approximately 4KB[2]. To figure out the relationship between initial congestion window, time and transmitted data, we assume the TCP connection is in slow start and without losses, thus the congestion window size in each RTT can be expressed as follow:

$$cwnd_i = init\_cwnd * 2^{i-1} \tag{1}$$

where $init\_cwnd$ is the initial congestion window. Therefore, the transmitted data in each RTT can be expressed by the following equation:

$$Trans_i = MSS * cwnd_i \tag{2}$$

$$= MSS * init\_cwnd * 2^{i-1} \tag{3}$$

where MSS is illustrated as maximum segment size in current TCP. By accumulating the data size transmitted in first i RTTs, the total data size can be expressed by:

$$TotalTrans_i = MSS * \sum_{t=1}^{t=i} init\_cwnd * 2^{t-1} \tag{4}$$

$$= MSS * init\_cwnd * (2^t - 1) \tag{5}$$

Thus, given a transmitted data size S, the data transmission latency $TransTime_S$ is:

$$TransTime_S = \lceil log_2(\frac{\lceil S/MSS \rceil}{init\_cwnd} + 1) \rceil * RTT \tag{6}$$

Finally, as mentioned in Section 2, it is obvious that reducing RTT and $init\_cwnd$ can reduce the data transmission latency, as well as HTTPS connection establishment time. The Fig. 3 illustrates the data transmission latency with different round-trip times and `init_cwnds` based on the equation (6). From the figure, shortening round-trip time is a better solution to reduce data transmission latency, which can be achieved by Edge Computing.

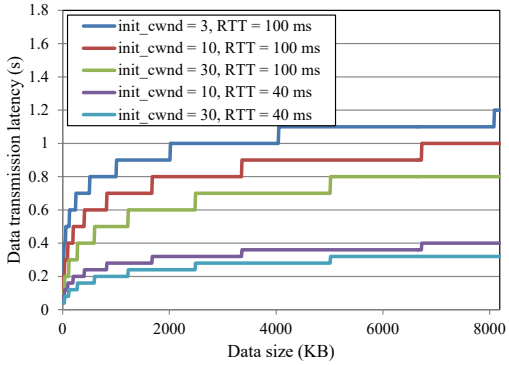


Figure 3. The CDF of data transmission latency with different initial congestion window and round-trip times.

Increasing the `init_cwnd` size also reduces the data transmission latency by reducing the round number of data transmission. For example, 90 packets (approximately 132 KB) cost 4 rounds and 2 rounds with different `init_cwnd` sizes of 10 and 30, respectively. Especially for these short data, lower than $init\_cwnd * MSS$ bytes, increasing congestion window size allows them to be transmitted in one RTT. The standardize TCP's initial congestion

1. There is a half overlap RTT between TCP handshaking and TLS handshaking, and we assume that RTT equals to around 50 ms.

2. We assume a maximum segment size of 1348 bytes that we evaluated based on our experimental environment.

window has been ten segments as the Internet Engineering Task Force accept the appeal from Google [13]. The default value of the `init_cwnd` on newer systems, *i.e.,* Linux 3.x -based, the Android and iOS, is 10, and the old one is 3 when MSS falls in between 1095 and 2190. Moreover, increasing initial congestion window can provide faster recovery from losses. For example, in TCP Reno version [18], a packet loss will lead to the cutting short of current congestion window size (*i.e.,* half), and an increased initial congestion window (at least 2x default value) can tolerate a packet loss in first round-trip time. After that, even though a packet loss happens, the connection will have the same congestion window size, at least, but have transmitted more data.

In this case, as a developer in the WoT era, there are two ways to improve the service quality here (*i.e.,* latency), which we will verify in Section 4: 1) pushing the computing at the edge of network, resulting in a lower round-trip time, which is what edge computing doing; and 2) adjusting the initial congestion window.

4. Experiment

In this section, we will introduce the testbed used in our experiments, and the experimental results, in terms of latency, RTT and retransmission rate, affected by moving speed, initial congestion window and server location.

4.1. Testbed

To measure how edge computing and increased initial congestion window affect the performance of data transmission via HTTPS, we have built a testbed for our experiments consisting of two virtual machines (VMs) with the vCPU (*i.e.,* Intel Xeon Broadwell E5-2680 v4 at 2.4GHz) on the Baidu Cloud Compute (BCC), located at Beijing and Suzhou, as well as the Raspberry Pi 3B with the Broadcom BCM2837 ARM A53 at 1.2GHz, equipped with 4G/LTE model (*i.e.,* Huawei ME909s-821) for communication, as shown in the Fig. 4. The HTTP service on the BCC simulates the data processing (*e.g.,* image recognition, image searching), and then respond with a URL that is used to assist in accessing the rich results. Also, a TLS termination proxy is provided by Nginx service to provide an HTTPS service interface for applications, located in the same VM with HTTP service, which handles incoming TLS connections, decrypt the TLS and pass on the unencrypted request to our HTTP service.

4.2. Experiment Setup

To evaluate impacts on data transmission latency, RTT and packet retransmission rate for uploading different sizes of data, caused by initial congestion window and server location, we designed three scenarios, including one static scenarios, two moving scenarios, in car with the speed of around 45 miles per hour (MPH) and in high-speed train with the speed of around 150 MPH, respectively. The servers

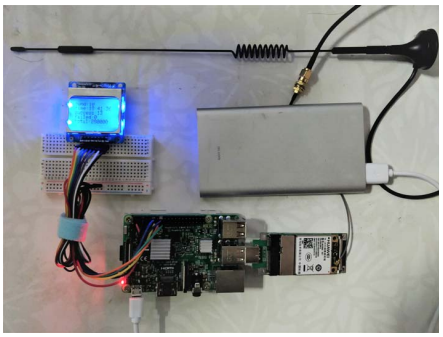


Figure 4. The Raspberry Pi testbed for our experiments.

providing HTTPS services locate at Suzhou (SZ in short) and Beijing (BJ in short), which are approximately 200 miles and 550 miles from Hefei, respectively. Thus, the former one would be an edge server, geographically, compared with the later one. In a static scenario, the experimental Raspberry Pi in Hefei will upload data to SZ's server and BJ's server with different data sizes and initial congestion windows. For the low-speed one of two moving scenarios, we drove a car in Hefei and took our experimental devices (*i.e.,* Raspberry Pi) to upload data to the server located at Suzhou. Regarding the high-speed one, we use Beijing's servers as the HTTP service providers since the experimental path of the train is closer to Beijing. Note that in moving scenarios, we mainly focus on the effect of different initial congestion windows, where the performance of using an edge server has been evaluated in the static scenario. For all experiments, the uploaded data sizes are from 10KB to 4MB, and the `init_cwnds` include the default value 10, in most of newer system, and the increased value 30. Note that we also increase the value of `init_rwnd` (receive window) to make sure the sender can send all packet in first RTT. All provided HTTP services will sleep one and half seconds once received the data, and then response the request with a short data size (*i.e.,* URL), to simulate the data processing (*e.g.,* image recognition, image searching). We captured all packets of clients using the `Tcpdump` tool.

4.3. Impact on RTT, Handshaking Time and Retransmission Rate

The average RTT is improved while using a nearer server. For example, as shown in the Table 2, the average RTT of Suzhou case is 37.82 ms, only 67% of the average RTT of Beijing case. Besides, the car case, having the same server location but a different speed to static Suzhou case, has a higher RTT, thus also a higher TCP handshaking time and a higher TLS handshaking time. Thus, a dynamical case will increase the RTT. It should be noted that here, we did not distinguish different initial congestion windows since it does not affect RTT.

The Fig. 5 shows the CDF of RTT in different scenarios. The analyzed results show that the RTT of the train case with

TABLE 2. RTT, TCP HANDSHAKING (TCP HS) TIME, TLS HANDSHAKING (TLS HS) TIME IN DIFFERENT SCENARIOS.

	Static SZ	Static BJ	Car	Train
RTT	37.82 ms	56.14 ms	40.82 ms	119.57 ms
TCP HS	40.99 ms	66.01 ms	58.71 ms	141.49 ms
TLS HS	147.57 ms	221.54 ms	161.71 ms	334.44 ms

speed of around 150 MPH is worst of these four cases, and 3% RTTs are higher than one second, in which the highest one is more than 20 seconds, exceeding the normal timeout value of an HTTP connection. The highest RTT in other three cases are lower than 1 second, 260 milliseconds, and 140 milliseconds for the car case, the static case of Beijing and the static case of Suzhou, respectively. Especially, the RTT of the car case is better than the one of the static case of Beijing.

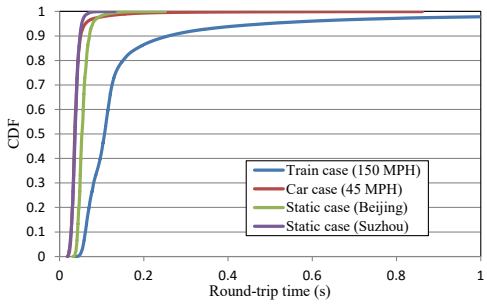
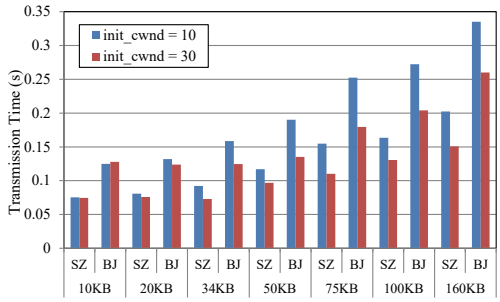


Figure 5. The CDF of round-trip time.

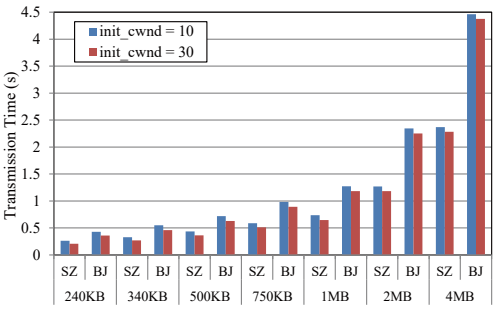
Besides, we also analyze the packet retransmission rate here. The packet retransmission rates in dynamical scenarios are larger than the static scenario. In the static scenario, it has hardly any packet retransmission, comparing to the dynamical scenarios, in which the packet retransmission rates are around 0.1% and 1%. In our observations on experimental traces of train case, some packet retransmissions happen in TCP handshaking phase, and it results in a low `init_cwnd`, *i.e.,* 3, for remained phases, such as TLS handshaking and data transmission, which will increase data transmission latency, significantly.

4.4. Impact on Data Transmission Latency

The static experiment results are depicted in the Fig. 6. As expected, the edge server (*i.e.,* Suzhou) performs a lower data transmission latency, and especially for the 4MB data, it is only half of the data transmission latency to Beijing. Moreover, increasing `init_cwnd` also reduces data transmission latency, particularly, it can reduce approximately 20% for these data from 30KB to 160KB in our experiments. The time on lower or larger data size is not improved. For a 1MB data, it also has an improvement of around 10%. Note that the reduced data transmission latency of large data volume is more than the one of small data volume,



(a) Data size from 10KB to 160KB.



(b) Data size from 240KB to 4MB.

Figure 6. Latency in a static scenario with different service locations and `init_cwnds`.

e.g., approximately 90ms for the 4MB case and 54ms for 240KB while the HTTPS service provider locates at Suzhou.

The Fig. 7 illustrates the results of the car case that has a speed of around 45 MPH in a city of China. We set up the results, evaluated in a static environment with a default `init_cwnd` (*i.e.,* 10), as the baseline. The figure shows a moving case has a lower network quality, resulting in a higher data transmission latency. Likewise, the increased `init_cwnd` can reduce the data transmission latency on a similar ratio.

The Fig. 8 shows the results of the high-speed train that has a higher speed than normal cars, *i.e.,* 150 MPH. In this case, the static case with a default `init_cwnd` (*i.e.,* 10) is set as the baseline. Similarly, the increased `init_cwnd` can reduce the data transmission latency, but with a higher ratio. For example, while it uploads 75KB and 100KB data, it will reduce up to 44% data transmission latency. The reason is that increasing initial congestion window allows faster recovery from losses.

In conclusion, the nearer service provider can reduce data transmission due to its lower RTT, which has been measured in Section 4.3, and it is more effective than increasing the initial congestion window. Besides, a larger congestion window size is effective in dynamical scenarios (*i.e.,* high-speed train).

5. Summary & Suggestion

The data size, uploaded by various things (*e.g.,* smart cameras, and other IoT devices) to Web-based services, are

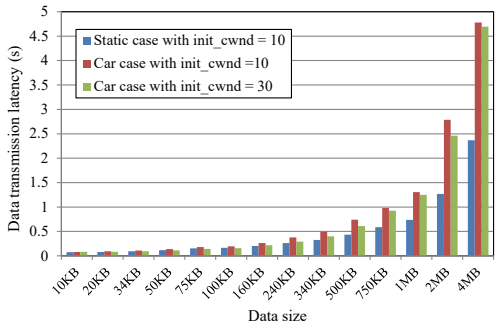


Figure 7. Latency in a car with different `init_cwnds`.

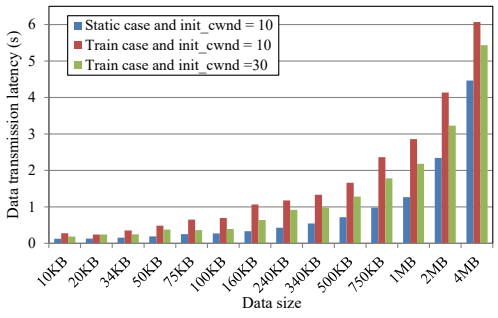


Figure 8. Latency in a high-speed train with different `init_cwnds`.

getting larger and larger, but they require a response in time from services hosted on WWW, *e.g.,* image recognition services. Based on our observation of a similar image searching service of Baidu, uploading data costs hundreds of milliseconds via a 4G cellular network, which is too large for some latency-intensive applications. Thus, we analyzed the procedure of that image service and found that using edge service providers and increasing initial congestion window can reduce the data transmission latency. Thus, we evaluated the performance with different locations and initial windows, as well as different speeds of users. The results show an edge service provider can provide a better RTT, resulting in a lower time consumption on handshaking and data transmission, and increased initial congestion window can reduce data transmission latency, especially in a dynamical scenario.

Thus, for these latency-intensive applications in WoT era, there are some ways to improve the performance: 1)offloading the data processing close to the data producers, such as a private cloud in local connected using a high speed network, or an edge/cloud server; and 2) setting a reasonable initial congestion window, which can upload the data in lesser RTTs. At last, we also notice that there are some other protocols over User Datagram Protocol (UDP), such as Quick UDP Internet Connections (QUIC) and KCP, KCP, a fast and reliable ARQ protocol. As a future work, we will measure and compare them, to figure out a guide for WoT developers.

References

[1] (2018) Cisco global cloud index: Forecast and methodology 2016-2021 white paper. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html

[2] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, *From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 97–129.

[3] (2018, Apr.) Amazon echo. [Online]. Available: https://developer.amazon.com/echo

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[7] X. Wang, A. Chowdhery, and M. Chiang, "Networked drone cameras for sports streaming," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 308–318.

[8] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image Recognition applications at the edge," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 17:1–17:13.

[9] D. Li, T. Salonidis, N. V. Desai, and M. C. Chuah, "Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2016, pp. 64–76.

[10] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 276–286.

[11] X. Nie, Y. Zhao, G. Chen, K. Sui, Y. Chen, D. Pei, M. Zhang, and J. Zhang, "Tcp wise: One initial congestion window is not enough," in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, Dec 2017, pp. 1–8.

[12] V. Jacobson, "Congestion avoidance and control," in *Symposium Proceedings on Communications Architectures and Protocols*, ser. SIGCOMM '88. New York, NY, USA: ACM, 1988, pp. 314–329.

[13] 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." *Computer Communication Review*, vol. 40, no. 3, pp. 26–33, 2010.

[14] (2018, Jan.) Wechat related network flow problem. [Online]. Available: http://weixin.qq.com/cgi-bin/readtemplate?nav=contact&t=weixin_faq_networkflow

[15] Q. Zhang, Z. Yu, W. Shi, and H. Zhong, "Demo abstract: Evaps: Edge video analysis for public safety," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2016, pp. 121–122.

[16] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.

[17] (2018, Jan.) Baidu image recognition: Recognize what's your see. [Online]. Available: http://image.baidu.com/?fr=shitu

[18] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling tcp reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, Apr 2000.