Polygon: A QUIC-Based CDN Server Selection System Supporting Multiple Resource Demands

Mengying Zhou[1,2], Tiancheng Guo[1,2], Yang Chen[1,2], Junjie Wan[3], Xin Wang[1,2]
[1]School of Computer Science, Fudan University, China
[2]Shanghai Key Lab of Intelligent Information Processing, Fudan University, China
[3]Huawei Technologies Co., Ltd., China
{myzhou19,tcguo20,chenyang,xinw}@fudan.edu.cn,wanjunjie2@huawei.com

ABSTRACT

CDN is a critical Internet infrastructure that helps Internet users get contents with a short delay. With the development of CDN application scenarios, CDN requests will involve more than one resource type. Unfortunately, the existing CDN server selection schemes targeting one resource type cannot select the most suitable CDN server by considering different resource types together. In this paper, we propose *Polygon*, a CDN server selection system supporting multiple resource demands. The keystone of Polygon is the deployment of a set of dispatchers at strategic network locations, which can be accessed via anycast. The dispatchers are responsible for resource status collection, server assignment with resource demands, and forwarding requests to suitable CDN servers. Meanwhile, Polygon adopts the 0-RTT and connection migration features of the QUIC protocol to mitigate the extra delay for connection and forwarding. We conduct real-world experiments on the Google Cloud Platform to demonstrate the advantages of Polygon. The results show that the deployment of the dispatchers enables Polygon to provide a better CDN service with a median job completion time reduction of up to 54.8%. Also, Polygon improves resource utilization efficiency by 13% in terms of bandwidth and by 7% in terms of CPU.

CCS CONCEPTS

• **Networks** → **Middle boxes / network appliances**; *Network resources allocation.*

KEYWORDS

CDN, QUIC, Anycast, Dispatchers, Overlay Network

ACM Reference Format:
Mengying Zhou[1,2], Tiancheng Guo[1,2], Yang Chen[1,2], Junjie Wan[3], Xin Wang[1,2]. 2021. Polygon: A QUIC-Based CDN Server Selection System Supporting Multiple Resource Demands. In *International Middleware Conference Industrial Track (Middleware Industry '21), December 6–10, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3491084.3491428

Middleware Industry '21, December 6–10, 2021, Virtual Event, Canada

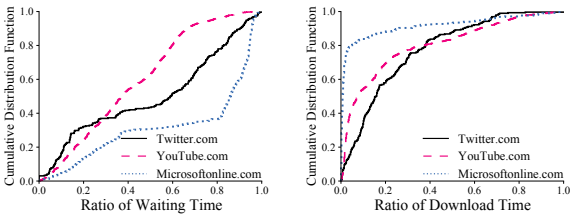
ACM ISBN 978-1-4503-9152-8/21/12…$15.00
https://doi.org/10.1145/3491084.3491428

1 INTRODUCTION

Content Delivery Network (CDN) is a critical Internet technology to provide quick responses to users' requests. By replicating contents, such as pictures, videos, databases, from the source server to a series of CDN servers worldwide, users can access these contents through nearby CDN servers. A reasonable allocation of replication CDN servers to users is the core assurance of CDN deployment. The traditional CDN server selection scheme is based on Domain Name Service (DNS) redirection [30], which is widely deployed by the largest CDN service providers such as Akamai. Recently, a series of anycast-based CDN solutions have been emerging. Anycast [31] allows mapping the same IP address to multiple servers and returning the server with the shortest routing path, which is naturally suitable for distributed networked systems like CDNs. However, anycast is not aware of the load status (e.g., CPU usage) of the CDN servers [8, 23]. To address this problem, there are several anycast-based CDN proposals realizing the load awareness [2, 3, 12]. One of them, named FastRoute [11], has been deployed in the production environment of the Bing search engine [6].

Unfortunately, these proposals still have the shortcoming of focusing on a single resource type, which brings the problem of ***incapable of selecting the most suitable CDN server by considering different resource types together.*** According to our motivating case study in §2, different CDN requests might require different types of resources. For instance, downloading large contents requires high bandwidth, while obtaining a set of small files is more concerned with the delay. Therefore, it is a multi-dimensional decision to determine the most suitable CDN server. In addition, single resource type-based methods would cause unequal server allocations, resulting in "hot zones" and resource idleness. Such ineffective resource allocation further significantly increases the cost of service providers.

In this paper, we propose ***Polygon***, a CDN server selection system supporting multiple resource demands while still benefiting from the advantages of anycast. The keystone of Polygon is the deployment of a set of dispatchers at strategic network locations. The dispatchers are responsible for resource status collection, server assignment with resource demands, and forwarding requests to suitable CDN servers. A dispatcher periodically collects resource status. Once receives the clients' requests, the dispatcher selects the appropriate servers based on the resource demands and the latest resource status. The introduction of the dispatchers will bring some additional overhead, especially extra delays for connecting and forwarding. To mitigate the side-effect, Polygon adopts Quick UDP Internet Connections (QUIC) [16, 21], a promising transmission



(a) The ratio of waiting time **(b) The ratio of download time**

Figure 1: The cumulative distribution functions of the ratios of waiting time and download time

Sensitivity Metric	Service Quality Level	RTT	Bandwidth	CPU
RTT	Poor	131 ms	248 Mb/s	1 standard vCPU
	Medium	112 ms	214 Mb/s	1 standard vCPU
	Good	96 ms	184 Mb/s	1 standard vCPU
Bandwidth	Poor	51 ms	3 Mb/s	1 standard vCPU
	Medium	34 ms	248 Mb/s	1 standard vCPU
	Good	11 ms	758 Mb/s	1 standard vCPU
CPU	Poor	27 ms	917 Mb/s	1 shared vCPU
	Medium	27 ms	917 Mb/s	1 standard vCPU
	Good	27 ms	917 Mb/s	4 standard vCPU

Table 1: The detailed server configurations on the testbed

protocol to achieve a low latency handshake and support connection migration. The 0-RTT [14] handshake feature of QUIC assists in establishing fast connections from the client to the dispatcher. Additionally, QUIC eliminates the re-connection process between the client and the server with connection migration features. The evaluation results show that the deployment of the dispatchers enables Polygon to achieve a better CDN service with a median job completion time reduction of up to 54.8%. Our contributions are summarized as follows:

- We introduce a motivating case study to demonstrate that in different application scenarios, users would have different resource type priorities when selecting CDN servers. We aim to support delay-sensitive, bandwidth-sensitive, and CPU-sensitive CDN server requests with an integrated solution.
- We propose Polygon, a QUIC-based CDN server selection system supporting multiple types of resource demands. Polygon also leverages the advantages of QUIC to tackle the extra delay and overhead caused by introducing the dispatchers.
- We conduct an evaluation of the real-world deployment on the Google Cloud Platform. The experimental results show that Polygon provides better CDN services with a median job completion time reduction of up to 54.8%. Polygon also improves resource utilization efficiency by 13% in terms of bandwidth and by 7% in terms of CPU.

2 MOTIVATING CASE STUDY

CDNs have been deployed for many years and support several content types, including web contents [40], video streaming [29] and replica databases [18, 41]. Generally, different types of CDN requests reveal distinct resource demands. Here, we introduce a case study to investigate the different CDN request patterns on three typical websites and verify that different requests are sensitive to different resource types.

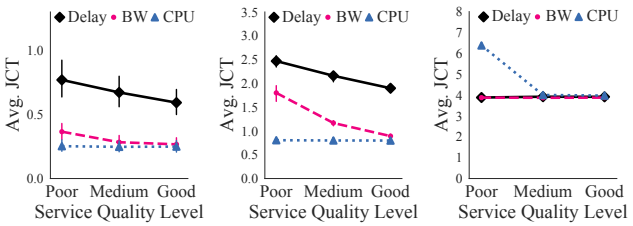
Three Types of CDN Request Patterns. Three typical websites `Twitter.com`, `YouTube.com` and `Microsoftonline.com` are chosen as cases, all of which have millions of users worldwide [15] and rely heavily on global CDN infrastructure [36]. We treat each requested file recorded by Chrome-HAR [7] as a CDN request. We study the ratios of waiting time and download time of CDN request completion time, respectively. In Fig. 1, we plot the cumulative distribution functions of the ratios of waiting time and download time on three websites. We notice that for `Microsoftonline.com`, more than 60% of the requests with ratios of waiting time more than 0.8. While `Twitter.com` and `YouTube.com` have fewer requests with such ratios of waiting time. Comparatively, the ratios of download time of the requests on `Microsoftonline.com` are shorter than those on `Twitter.com` and `YouTube.com`.

The above case study shows that the ratios of waiting time and download time for completing each request for the three typical web services are quite different. According to this phenomenon, we intuitively assume that CDN content requests have different sensitivity priorities to different resource types in different application scenarios. By carefully considering a set of classic application scenarios, we divide CDN content requests into three categories that are sensitive to different resource types, i.e., delay, bandwidth, and CPU capability.

- **Delay-Sensitive** requests are sensitive to network delay. This type of sensitivity generally occurs in *web browsing*, which needs to fetch a number of small-size contents from web pages.
- **Bandwidth-Sensitive** requests are sensitive to available bandwidth. This type of sensitivity is highly related to *content downloads*, which includes downloading large files and video streaming.
- **CPU-Sensitive** requests are sensitive to CPU capability. This type of sensitivity often occurs in *computing tasks*, mainly for database queries, which require high I/O and intensive computation.

Sensitivity to Different Recourse Types. We design a test on the Google Cloud Platform to verify that the three CDN requests are sensitive to different resource types. We use "poor", "medium", and "good" to represent the servers' various service quality levels of delay, bandwidth, and CPU. Specifically, we use Round-Trip Time (RTT) to quantify the network delay. The detailed server configurations are listed in Table 1. Then we use the same client to successively make the following three types of requests to these servers. For delay-sensitive requests, we crawl the frontpages of Alexa Top 500 Sites for our study and generate visits to these crawled webpages. For bandwidth-sensitive requests, we use a video with a size of 5MB to generate a media CDN request and visit it for 10 times. For CPU-sensitive requests, we perform 100 random queries in a database with one million entries as computational requests.

We record the job completion time (JCT) of the above three requests on each server listed in Table 1. JCT is defined as the time to complete a CDN request task. The results in Fig. 2 demonstrate that there are obvious differences in the sensitivity of the three types of



(a) Delay-sensitive (b) Bandwidth-sensitive (c) CPU-sensitive

Figure 2: The average JCT of the three CDN request types

CDN requests to different resource types. Fig. 2(a) shows that with the increase of the delay service quality level, the average JCT of delay-sensitive requests reduces significantly. However, it does not change much with the fluctuation of other resources. Similarly, in Fig. 2(b) and Fig. 2(c), the average JCT values of bandwidth-sensitive and CPU-sensitive requests rapidly decline with the upgrade of the capability of corresponding resource types. In comparison, the average JCT is relatively stable with changes in the irrelevant resource types. Therefore, to achieve a better CDN service performance, we need to consider different resource types jointly instead of a single resource type.

3 DESIGN FOR POLYGON

In this section, we present the design of our proposal, Polygon, a CDN server selection system supporting multiple resource demands. First, we present the overall workflow of our architecture (§3.1). Subsequently, we list some practical challenges in implementing our system (§3.2). Furthermore, we introduce how we address each practical challenge, including resource status collection (§3.3), server assignment with resource demands (§3.4), and request forwarding to realize content transmission (§3.5).

3.1 Design Overview

Inspired by our findings in §2, it is promising to build a CDN server selection system by considering the demands of multiple resource types jointly. We believe it will be more useful than existing CDN server selection schemes, which only consider the topological closeness of the Internet [2, 8, 23]. We propose Polygon, a QUIC-based CDN server selection system supporting multiple types of resource demands. Similar to [22], we deploy a set of dispatchers in the network, which are responsible for resource status collection, server assignment with resource demands, and forwarding clients' requests to the suitable CDN servers. As shown in Fig. 3, the workflow of Polygon is as follows:

Step 1: Collecting Real-Time Resource Status. To allocate CDN servers based on resource demands, Polygon needs to periodically collect resource status and store them in dispatchers. For the resource status related to the network, e.g., delay and bandwidth, Polygon estimates the status in an aggregative way. By dividing the Internet into several regions, Polygon deploys a dispatcher in each region and obtains the representative network status by the measurement between the server and the dispatcher. For other resources, such as the CPU capability, servers directly send the status to the dispatcher. (§3.3)

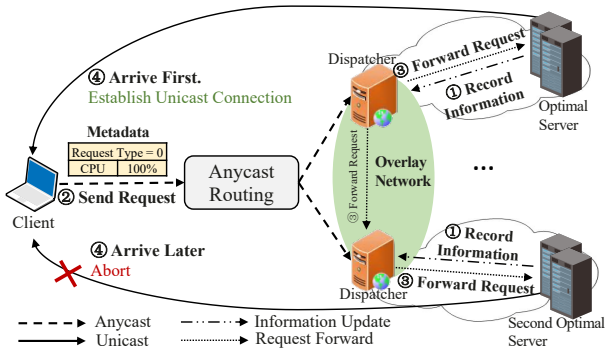


Figure 3: The workflow of Polygon for CDN server assignment

Step 2: Sending Request and Selecting Server. Then, the client sends the request with the resource demands and waits for the CDN server assignment. Polygon supports both the pre-defined classic resource compositions and customizable demand weights with the metadata block in QUIC. Moreover, to handle the failed responses, Polygon introduces a Demand Restriction Allocation (DRA) algorithm with the redundant forwarding mechanism when assigning suitable CDN servers (§3.4).

Step 3: Forwarding Request to Suitable Server. Polygon makes use of the advantages of QUIC to avoid unnecessary delay. In §3.5, we take the 0-RTT handshake feature in QUIC to establish fast connections. Furthermore, we build an overlay network to ensure quick forwarding among dispatchers, and eliminate re-connection delay between a client and a server by leveraging connection migration features of QUIC.

Step 4: Establishing Unicast Connection and Content Transmission. Considering the unicast connection can achieve a more stable content transmission than the anycast connection [1], Polygon will establish a unicast connection between client and server based on the unicast server address in the response packet.

3.2 Practical Challenges

However, there are several practical challenges as follows when implementing Polygon.

Diversity of Resource Status. Server-client node pairs are numerous on the Internet, and it is impossible to measure the end-to-end links between all these pairs in a reasonable time frame. Also, given the dynamic of the Internet, the resource status may be quite different between adjacent moments. Therefore, it is impossible for the CDN selection system to measure and record the resource status of all server-client pairs using only one round of measurement.

Effective Demand Delivery and Selection Algorithm. Since Polygon will support both classic resource demands and customizable resource demands, we are faced with the design challenge of realizing these two functions effectively. Concomitantly, the CDN allocation algorithm should be flexible under multiple resource types and robust with dynamic network environments.

Extra Delay for Connecting and Forwarding. In Polygon, the connection establishment between a client and a server has two steps. A dispatcher needs to perform a CDN assignment first. Then the client establishes another connection with the selected CDN server, which will cause extra connection delay. Reducing the extra

connection delay is the key to keeping the advantages of our CDN selection system.

3.3 Collection of Resource Status

As mentioned before, the biggest problem of collecting resource status is the diversity of different server-client pairs. To solve this problem, we obtain the information of network resources between the clients and the servers in an aggregative way.

We deploy Polygon globally. There is one dispatcher for resource status collection in each region. We can define a region as a geographically adjacent area (e.g., a state, province, or city) or a network zone (e.g., an autonomous system). In this paper, we consider three representative service resource types when requesting CDN contents, i.e., delay, bandwidth, and CPU capability.

Delay. Hosts that are geographically close will have similar network environments [24]. Network delay is highly correlated with the link hops and geographic distance [5]. Since Polygon deploys a series of dispatchers to collect the resource status worldwide, we can leverage the dispatcher in each region as the representative agent to provide the network delay measurement. As in Table 1, we use RTT to quantify the network delay. The RTT is obtained by `Ping` [28]. Polygon will record the RTT between every server-dispatcher pair by conducting probing every hour since the RTT is relatively stable. The results are stored by MySQL Database.

Bandwidth is also a type of network resource strongly related to geographic location because cross-region bandwidth is much more scarce than the intra-region bandwidth (5-20× lower) [34]. Similar to the RTT measurement, we use the dispatcher in each region as the representative agent to estimate the available bandwidth. We use the method proposed in [35] to measure the available bandwidth. However, bandwidth measurement needs to transmit more data than measuring the delay. Taking into account both the timeliness and overhead of the measurement, we set 1.5 seconds as the interval to measure the available bandwidth and store it in Redis Database.

CPU Capability is a type of resource that can be measured by the server itself. We obtain the available CPU capability by the calculation of *idle rate* × *number of CPU cores* × *CPU clock frequency*. Similar to bandwidth consideration, the measurement duration of the CPU is set at 1.5 seconds. The measurement results are stored by Redis Database.

3.4 Metadata and Server Selection Algorithm

After collecting real-time resource status, another challenge is how to attach the resource demand to the data packet and design a robust selection algorithm. Polygon provides both the pre-defined classic resource compositions and customizable demand weights with the metadata block in QUIC. We also implement a DRA algorithm with redundant forwarding to handle the single point of failure.

Metadata in QUIC. In QUIC, there is an extended metadata block at the handshake packet that can be customized [16]. We append the specific resource demands with the metadata, and the structure of metadata is shown in Fig. 4. The metadata can be divided into three parts: the requested resource composition, the specific CDN resource request flag, and the weight of each resource type demand.

For the requested resource composition part, we pre-define a set of resource compositions by assigning each type a positive integer. For each resource composition, we pre-define the demand of different resource types. Each classic resource composition (e.g., bandwidth-sensitive file downloading) has been pre-configured, and the dispatchers are aware of the detailed demand of different resource types.

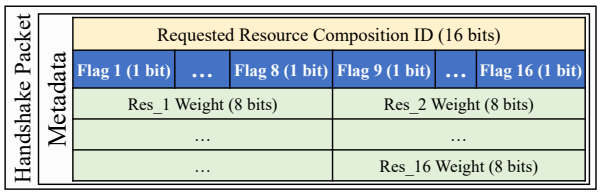


Figure 4: The design of metadata in QUIC. The metadata not only provides pre-defined classic resource compositions, but also allows a client to set customizable demand weights.

We also allow a client to have customized resource demands. To achieve this, the requested resource composition ID should be set as 0. In addition, we apply a 16-bit flag field and a 128-bit weight field to specify detailed resource demands. Each bit of the 16-bit flag field corresponds to a given resource type, and therefore this flag field can cover the commonly used resource types. For each demand, we use 8 bits as the weight of each resource type. Considering the transmission efficiency, the client just transmits the weight block of the resource type with the TRUE flag. The dispatcher can decode the weight information according to the binary information of the flag in the metadata block.

Demand Restriction Allocation (DRA) Algorithm. The core of the DRA algorithm is the server resource scoring operation. For each candidate server, we calculate the server score according to the resource flags and corresponding weights from metadata. A larger server score indicates that this server is more suitable to serve the request. Two aspects determine the server score. The first part is the amount of currently available resources, the most intuitive metric indicating the availability. The second part is calculated based on the maximum capability of resources since more powerful resource could provide better services. We calculate the sum of the weights of existing connections and the weight of the to-be-established connection. Then the maximum capability of the resource is divided by this sum to obtain the average availability. For each type of resource, we add the scores of the two aspects, and then multiply it by the weight of the corresponding resource type. Finally, the complete server score is the cumulative value of each resource score.

Redundant Forwarding. To avoid the assignment mistakes caused by potential sharp capacity degradation of the optimal node, we introduce a redundant forwarding mechanism. Ideally, Polygon will pick up both the optimal and the second optimal servers according to the server scores and forward requests to both of them. In practice, only when the second optimal server's score is 10% less than the optimal server's and the RTT value is shorter than 30ms, we will trigger the redundant forwarding, which avoids the disadvantages caused by the second optimal server.

Owing to redundant forwarding, the client might receive two servers' responses successively. The client will only respond to the first received response and establish a unicast connection with the corresponding server, while discarding another server's response.

3.5 Request Forwarding by Dispatchers

The extra delay introduced by Polygon includes the delay of reaching the dispatcher, request forwarding, and re-establishing the connection between the client and the CDN server. The key to maintaining the advantages of the CDN selection system is to eliminate these delays.

Polygon makes use of the advantages of QUIC to solve this problem. QUIC is proposed on top of UDP to enhance the efficiency of data transmission [21]. Compared with TCP, QUIC introduces many new features. For example, it achieves a lower latency handshake with 1-RTT and 0-RTT, and supports connection migration to transfer from one server to another smoothly.

Zero-Latency Connection Establishment to Dispatchers. Polygon uses anycast to access the nearest dispatcher and leverages the 0-RTT feature of QUIC to reduce the handshake delay. When the client has ever connected to the dispatcher, the shared key between the client and the dispatcher will be kept, and this key will be reused in the next connection. In other words, the client can directly transmit the contents without the handshake process, which is known as the 0-RTT handshake. Frequent interactions between the client and the dispatcher provide an opportunity for the 0-RTT handshake, which vastly decreases the connection time.

Fast Forwarding via Overlay Network. The forwarding delay is acceptable for the servers that reside in the same region of a dispatcher since they are close to each other and the intra-region delay is short. However, there are some situations where we need to forward requests to servers in other regions, which will result in high delays [37]. Therefore, we construct a fast-forwarding overlay network [22, 33] to connect the dispatchers. Via this overlay network, data packets can quickly reach another region through a GRE tunnel [10]. Then the packets will be sent to the final server in the same region through low-latency forwarding.

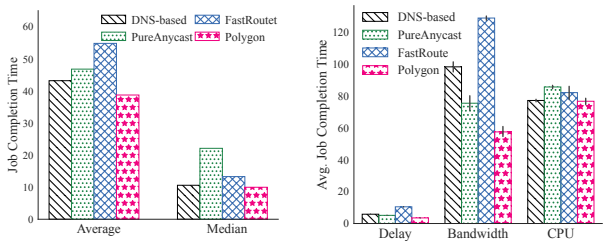
Eliminating Re-Connection between Client and Server. Finally, Polygon leverages the connection migration function in QUIC to eliminate the re-connection operation between the client and the server. Connection migration realizes an unconscious transfer from one node to another. Universally Unique Identifier (UUID), as a unique identifier of the connection instead of a 5-tuple in QUIC, guarantees the feasibility of the continuous connection even when the IP address changes. Therefore, the server can continue the subsequent data transmission instead of re-establishing a new connection.

The server will append its unicast address in the *preferred_address (0x000d)* block of the handshake response packet. In this way, when the client receives the response packet, the client can directly establish the subsequent unicast connection with the unicast address. With connection migration, Polygon can accomplish sending requests, forwarding requests, and establishing data transmission connections through only one QUIC connection, which greatly reduces the delay.

4 EVALUATION

In this section, we introduce our evaluation and advantages of Polygon in terms of JCT and resource utilization efficiency.

Baselines. We select three representative CDN server allocation systems as the baselines and compare Polygon with them. The first is the widely used DNS-based CDN selection system [30], which has been applied by mainstream CDNs such as Akamai. The rest two are both built on the anycast routing mechanism. One is PureAnycast [4], a naive anycast CDN selection system, and the other is FastRoute [11], a CDN selection adopted by Bing [6]. FastRoute arranges CDN servers in hierarchical anycast layers for offloading traffic under heavy load. To avoid bias by adopting different transmission protocols, we implement the baselines with the QUIC protocol. Thus we could ensure a fair comparison.



(a) The average and median values of job completion time **(b) The average JCT of requests sensitive to different resource types**

Figure 5: The JCT performance comparison

Evaluation Configuration. We evaluate the three baselines and Polygon on the Google Cloud Platform. We create five CDN servers located in five continents, i.e., Asia, Australia, Europe, North America, and South America. To implement Polygon, we deploy one dispatcher in each continent. In particular, FastRoute needs to build a virtual hierarchical architecture. We select the servers located in Asia, Europe, and North America to form the outer layer and the servers in Australia and South America to constitute the standby servers in the inner layer. The anycast IP address is provided by the load balancing service on the Google Cloud Platform [25]. The clients are deployed on virtual machines in 10 Google Cloud data centers. Each client executes ten processes simultaneously to emulate a group of clients. We arrange three clients in Asia, three clients in North America, two clients in Europe, one client in Australia, and one client in South America. We construct the evaluated requests by randomly selecting the three types of requests in §2 with the ratio of 4:4:1. All machines (including clients, servers, dispatchers) share the same machine setting, i.e., any of which runs Ubuntu 18.04 LTS and is configured with one standard vCPU and 3.75 GB memory.

Less Job Completion Time. Job completion time is a key metric for measuring the performance of CDN content fetching [19, 39]. JCT is defined as the time to complete a CDN request task. In Fig. 5(a), we plot the average and median values of JCT of the four systems. It can be found that Polygon outperforms all baseline schemes and achieves less JCT in terms of either the average or median value. Polygon has a 29.3% reduction in average JCT compared with FastRoute, and 5.8% from the DNS-based scheme. For the median JCT, Polygon has a 54.8% reduction from PureAnycast and a 3.1% reduction from the DNS-based scheme.

Further, in Fig. 5(b), we investigate what types of requests promote Polygon's advantages. Polygon achieves a smaller JCT value for each type of request. Especially for bandwidth-sensitive requests, Polygon displays a 55.4% reduction, i.e., about 71.4 ms

Method	# BW Req.	BW Cost / # BW Req.	# CPU Req.	CPU Cost / # CPU Req.
DNS-based	1570	7.04	421	0.74
PureAnycast	1915	6.31	576	0.60
FastRoute	497	12.56	380	0.90
Polygon	***2166***	***4.71***	***619***	***0.49***

Table 2: The number of requests completed and the resource cost of each request. BW is an abbreviation for bandwidth.

shorter compared with the PureAnycast scheme. Polygon achieves such improvement by assisting the client in getting rid of crowded servers and flexibly choosing the CDN server that matches the client's resource needs in other regions. 64% of CPU-sensitive requests and 34% of bandwidth-sensitive requests are redirected to other regions. When a nearby CDN server reaches its capacity, Polygon forwards the request to other less crowded servers to improve file transmission efficiency. Meanwhile, since subsequent requests are offloaded to other regions, the performance of the existing tasks would not be further downgraded. Overall, Polygon effectively serves the users with different resource demands by considering different types of resources together.

Higher Resource Utilization Efficiency. In addition to reducing the JCT, Polygon can also improve the overall resource utilization efficiency on the server-side and further cut down the cost for service providers [26]. In Table 2, we list the number of requests completed in the same 2 hours and the resource cost to complete per request [29]. For example, the CPU resource cost for each request is calculated by dividing the CPU usage on all links by the number of requests. The calculation method for the bandwidth resource is similar. Polygon outperforms baselines for both CPU-sensitive requests and bandwidth-sensitive requests. Compared with the suboptimal PureAnycast, Polygon increases the bandwidth-sensitive requests throughput by 13% and decreases the cost by 25%. While the CPU-sensitive request throughput is lifted by 7%, and the cost is reduced by 18%. By forwarding the crowded regions' requests, Polygon makes better use of the unoccupied servers and alleviates the resource preemption in these crowded regions. Due to the unbalanced global traffic distribution [17], the DNS-based scheme and PureAnycast cannot respond quickly with the limited resources of crowded regions, resulting in fewer completed requests. In particular, the FastRoute scheme performs worse than other schemes since all requests are concentrated on a few servers owing to its layer-by-layer activation structure.

5 RELATED WORK

Anycast-Based CDN. Anycast is a classic technology used in modern CDNs. It is naturally suitable for large-scale Internet content delivery since it matches the core idea of CDN, which is to place resources on nodes near users. Flavel et al. [11] proposed FastRoute built on anycast to allow users to access the closest services. They deployed FastRoute on the Bing search engine and conducted a large-scale performance measurement [6]. However, they found that although anycast shows good performance for CDN server selection, there existed the problem of control loss [23] that FastRoute guides approximately 20% of requests to a second optimal CDN endpoint. For this problem, Alzoubi et al. [2, 3] presented a load-aware anycast CDN architecture by using server and network load feedback to control the CDN server redirection. Fu et al. [12] proposed T-SAC to achieve a precise control for fine-grained traffic using a single 1-bit non-redirection flag. In addition, to solve the possible connection interruption problem with anycast, Lai and Fu [20] suggested to direct the anycast traffic to their in-the-middle MIMA nodes and convert anycast CDN target addresses to unicast addresses.

Load Balancing. Load balancing is a critical component in various Internet-scale distributed systems. Ananta [32] was introduced by Patel et al. in 2013, and Eisenbud et al. proposed Maglev [9] in 2016. These two load balancing systems are deployed on the large-scale infrastructure of Microsoft and Google, respectively. Apart from traffic-aware load balancing studies, there were some load balancing work for other considerations. Considering the huge energy consumption, Mathew et al. [26] took energy optimization as the primary principle and designed an energy-aware optimization algorithm. Zhang et al. [38] proposed a load balancing scheme for scenarios under uncertainties and achieved a significant improvement when the switches occasionally failed. Miao et al. [27] leveraged switching ASICs to build faster load balancers, which were able to handle 10 million connections simultaneously. While Gandhi et al. [13] embedded the load balancing function into hardware switches, which had 10 times more capacity and 10 times less latency than software-based solutions.

6 CONCLUSION

In this paper, we propose Polygon, a CDN server selection system supporting multiple types of resource demands by leveraging the advantages of both QUIC and anycast. With the help of a set of in-network dispatchers, Polygon can select suitable CDN servers based on the resource demand and resource availability. Owing to the 0-RTT and connection migration features of QUIC, Polygon can establish fast connections from a client to a dispatcher and eliminate the re-connection process between a pair of client and server. Furthermore, to mitigate the request forwarding delay across different regions, Polygon realizes quick forwarding by building an overlay network between the dispatchers. Our evaluation shows that Polygon reduces the median JCT by up to 54.8% compared with the baseline schemes. Additionally, Polygon reduces resource idleness and achieves more effective resource utilization by forwarding the requests across different regions.

In the future, we plan to have a wider range of practical deployment of Polygon, and conduct more comprehensive measurement studies. Meanwhile, we will further explore and address the potential scalability and dynamic challenges of applying Polygon in different kinds of large-scale real-world network applications.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No. 61971145) and the HUAWEI research collaboration YBN2019125184. Yang Chen is the corresponding author.

REFERENCES

[1] Zakaria Al-Qudah, Seungjoon Lee, Michael Rabinovich, Oliver Spatscheck, and Jacobus Van Der Merwe. 2009. Anycast-aware transport for content delivery networks. In *Proc. of WWW*.

[2] Hussein A. Alzoubi, Seungjoon Lee, Michael Rabinovich, Oliver Spatscheck, and Jacobus Van Der Merwe. 2011. A Practical Architecture for an Anycast CDN. *ACM Transactions on the Web* 5, 4 (2011), 1–29.

[3] Hussein A. Alzoubi, Lee Seungjoon, Michael Rabinovich, Oliver Spatscheck, and Jacobus Van Der Merwe. 2008. Anycast CDNs revisited. In *Proc. of WWW*.

[4] Abbie Barbir, Brad Cain, Raj Nair, and Oliver Spatscheck. 2003. Known content network (CN) request-routing mechanisms. RFC 3568. Available: https://www.rfc-editor.org/rfc/rfc3568.html. Accessed: 2021-10-10.

[5] CJ Bovy, HT Mertodimedjo, Gerard Hooghiemstra, Henk Uijterwaal, and Piet Van Mieghem. 2002. Analysis of end-to-end delay measurements in Internet. In *Proc. of the PAM*.

[6] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. 2015. Analyzing the performance of an anycast CDN. In *Proc. of IMC*.

[7] Andrea Cardaci. 2017. Chrome HAR Capturer. https://github.com/cyrus-and/chrome-har-capturer. Accessed: 2021-10-10.

[8] Danilo Cicalese, Jordan Augé, Diana Joumblatt, Timur Friedman, and Dario Rossi. 2015. Characterizing IPv4 anycast adoption and deployment. In *Proc. of CoNEXT*.

[9] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A fast and reliable software network load balancer. In *Proc. of NSDI*.

[10] Dino Farinacci, Tony Li, Stan Hanks, David Meyer, and Paul Traina. 2000. Generic routing encapsulation (GRE). RFC 2784. Available: https://www.rfc-editor.org/rfc/rfc2784.html. Accessed: 2021-10-10.

[11] Ashley Flavel, Pradeepkumar Mani, David A. Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. 2015. FastRoute: A scalable load-aware anycast routing architecture for modern CDNs. In *Proc. of NSDI*.

[12] Qiang Fu, Bradley Rutter, Hao Li, Peng Zhang, Chengchen Hu, Tian Pan, Zhangqin Huang, and Yibin Hou. 2018. Taming the Wild: A Scalable Anycast-Based CDN Architecture (T-SAC). *IEEE Journal on Selected Areas in Communications* 36, 12 (2018), 2757–2774.

[13] Rohan Gandhi, Hongqiang Harry Liu, Y. Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. 2015. Duet: Cloud scale load balancing with hardware and software. *Computer Communication Review* 44, 4 (2015), 27–38.

[14] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 2017. 0-RTT key exchange with full forward secrecy. In *Proc. of Eurocrypt*.

[15] Alexa Internet. 2021. The top 500 sites on the web. https://www.alexa.com/topsites. Accessed: 2021-10-10.

[16] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-based multiplexed and secure transport. RFC 9000. Available: https://www.rfc-editor.org/rfc/rfc9000.html. Accessed: 2021-10-10.

[17] Anish Jindal, Gagangeet Singh Aujla, Neeraj Kumar, Rajat Chaudhary, Mohammad S Obaidat, and Ilsun You. 2018. SeDaTiVe: SDN-enabled deep learning architecture for network traffic control in vehicular cyber-physical systems. *IEEE Network* 32, 6 (2018), 66–73.

[18] Nattiya Khaitiyakun and Teerapat Sanguankotchakorn. 2014. An analysis of data dissemination on VANET by using content delivery network (CDN) technique. In *Proc. of AINTEC*.

[19] Fan Lai, Jie You, Xiangfeng Zhu, Harsha V Madhyastha, Implementation Nsdi, Fan Lai, Jie You, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2020. Sol: Fast Distributed Computation Over Slow Networks. In *Proc. of NSDI*.

[20] Jeffrey Lai and Qiang Fu. 2016. Man-In-the-Middle Anycast (MIMA): CDN User-Server Assignment Becomes Flexible. In *Proc. of LCN*.

[21] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-scale Deployment. In *Proc. of SIGCOMM*.

[22] Xuebing Li, Bingyang Liu, Yang Chen, Yu Xiao, Jiaxin Tang, and Xin Wang. 2019. Artemis: A practical low-latency naming and routing system. In *Proc. of ICPP*.

[23] Zhihao Li, Neil Spring, Dave Levin, and Bobby Bhattacharjee. 2018. Internet anycast: Performance, problems, & potential. In *Proc. of SIGCOMM*.

[24] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. 2013. Stronger semantics for low-latency geo-replicated storage. In *Proc. of NSDI*.

[25] Google Co., Ltd. 2021. Google Cloud Load Balancing. Available: https://cloud.google.com/load-balancing. Accessed: 2021-10-10.

[26] Vimal Mathew, Ramesh K. Sitaraman, and Prashant Shenoy. 2012. Energy-aware load balancing in content delivery networks. In *Proc. of INFOCOM*.

[27] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs. In *Proc. of SIGCOMM*.

[28] David L. Mills. 1983. Internet delay experiments. RFC 0889. Available: https://www.rfc-editor.org/rfc/rfc0889.html. Accessed: 2021-10-10.

[29] Matthew K. Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. 2015. Practical, Real-time Centralized Control for CDN-based Live Video Delivery. In *Proc. of SIGCOMM*.

[30] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. 2010. The Akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review* 44, 3 (2010), 2–19.

[31] Craig Partridge, Trevor Mendez, and Walter Milliken. 1993. Host anycasting service. RFC 1546. Available: https://www.rfc-editor.org/rfc/rfc1546.html. Accessed: 2021-10-10.

[32] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A. Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, Changhoon Kim, and Naveen Karri. 2013. Ananta: Cloud scale load balancing. In *Proc. of SIGCOMM*.

[33] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, and Pravin Shelar. 2015. The design and implementation of Open vSwitch. In *Proc. of NSDI*.

[34] Zhirong Shen and Patrick PC Lee. 2018. Cross-rack-aware updates in erasure-coded data centers. In *Proc. of ICPP*.

[35] Jacob Strauss, Dina Katabi, and Frans Kaashoek. 2003. A Measurement Study of Available Bandwidth Estimation Tools. In *Proc. of IMC*.

[36] Ruben Torres, Alessandro Finamore, Jin Ryong Kim, Marco Mellia, Maurizio M. Munafò, and Sanjay Rao. 2011. Dissecting video server selection strategies in the YouTube CDN. In *Proc. of ICDCS*.

[37] Gaoxiong Zeng, Wei Bai, Ge Chen, Kai Chen, Dongsu Han, Yibo Zhu, and Lei Cui. 2019. Congestion control for cross-datacenter networks. In *Proc. of ICNP*.

[38] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient datacenter load balancing in the wild. In *Proc. of SIGCOMM*.

[39] Jia Zhang, Enhuan Dong, Zili Meng, Yuan Yang, Mingwei Xu, Sijie Yang, Miao Zhang, and Yang Yue. 2021. WiseTrans: Adaptive Transport Protocol Selection for Mobile Web Service. In *Proc. of WWW*.

[40] Liang Zhang, Fangfei Zhou, Alan Mislove, and Ravi Sundaram. 2013. Maygh: Building a CDN from client web browsers. In *Proc. of EuroSys*.

[41] Wei Zhang, Zhihui Lu, Ziyan Wu, Jie Wu, Huanying Zou, and Shalin Huang. 2018. Toy-IoT-Oriented data-driven CDN performance evaluation model with deep learning. *Journal of Systems Architecture* 88 (2018), 13–22.