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

Mengying Zhou, *Graduate Student Member, IEEE*, Tiancheng Guo, Yang Chen, *Senior Member, IEEE*, Yupeng Li, *Member, IEEE*, Meng Niu, *Member, IEEE*, Xin Wang, *Member, IEEE*, and Pan Hui, *Fellow, IEEE*

***Abstract*— CDN is a crucial Internet infrastructure ensuring quick access to Internet content. With the expansion of CDN scenarios, beyond delay, resource types like bandwidth and CPU are also important for CDN performance. Our measurements highlight the distinct impacts of various resource types on different CDN requests. Unfortunately, mainstream CDN server selection schemes only consider a single resource type and are unable to choose the most suitable servers when faced with diverse resource types. To fill this gap, we propose *Polygon*, a QUIC-powered CDN server selection system that is aware of multiple resource demands. Being an advanced transport layer protocol, QUIC equips Polygon with customizable transport parameters to enable the seamless handling of resource requirements in requests. Its 0-RTT and connection migration mechanisms are also utilized to minimize delays in connection and forwarding. A set of collaborative measurement probes and dispatchers are designed to support Polygon, being responsible for capturing various resource information and forwarding requests to suitable CDN servers. Real-world evaluations on the Google Cloud Platform and extensive simulations demonstrate Polygon's ability to enhance QoE and optimize resource utilization. The results show up to a 54.8% reduction in job completion time, and resource utilization improvements of 13% in bandwidth and 7% in CPU.**

***Index Terms*— CDN, QUIC, resource allocation, dispatcher, overlay network, anycast.**

## I. Introduction

CONTENT Delivery Network (CDN) is a vital Internet technology that quickly delivers various content to users. By replicating content from the source server to CDN servers worldwide, users can access content through nearby servers. Appropriate assignment of CDN servers to users [1], [2], [3] is essential for ensuring CDN service quality. Currently, there are two types of widely used CDN server selection methods. One uses the Domain Name System (DNS) to locate servers with the shortest Round-Trip Time (RTT) [4], adopted by commercial CDN providers like Akamai, Fastly, and EdgeCast. The other solution is based on anycast routing [5], [6], [7], [8]. Anycast [9] allows mapping the same IP address to multiple servers and routing to the servers with the shortest network hops according to routing protocols, making it well-suited for CDNs. Among them, FastRoute [8], which realizes CPU load awareness, has been deployed in Microsoft's Bing search engine [10].

However, these schemes have the drawback of considering only one single resource type, resulting in the allocation of unsuitable CDN servers when multiple resource types are required [2], [3]. As described in our motivating case study in Section II, different CDN requests may necessitate different resource types. For example, downloading large content requires high bandwidth, while obtaining a set of small files prioritizes low latency. Moreover, methods based on single resource types are vulnerable to population distribution, leading to hot zone problems [11] and inefficient resource utilization in uncrowded areas, significantly increasing service providers' cost [12].

To address this gap, we propose ***Polygon***, an efficient and scalable CDN server selection system supporting multiple resource requirements. Polygon is built on QUIC [13], [14], an emerging transport layer protocol, utilizing its customizable parameters to transmit resource demand information. Equipped with a set of dispatchers, Polygon parses the resource information in CDN requests. Then, using real-time resource status collected by measurement probes, it identifies the appropriate CDN servers and forwards the requests accordingly. Introducing dispatchers could bring extra delays in connection and request forwarding. To mitigate such delays, Polygon leverages QUIC's 0-RTT handshake [15] and connection migration mechanisms to minimize connection delays between the client, dispatcher, and server. We conduct a real-world evaluation on the Google Cloud Platform. Compared with state-of-the-art solutions, Polygon improves CDN performance with a median job completion time reduction of up to 54.8%. Polygon also increases bandwidth utilization by 13% and CPU utilization by 7%. Further extensive simulations demonstrate

Manuscript received 25 October 2022; revised 30 May 2023, 13 October 2023, 4 February 2024, and 1 April 2024; accepted 20 May 2024; approved by IEEE/ACM Transactions on Networking Editor K. Park. This work was supported in part by the National Natural Science Foundation of China under Grant 61971145 and in part by the HUAWEI Research Collaboration under Grant YBN201912518. *(Corresponding author: Yang Chen.)*

Mengying Zhou, Tiancheng Guo, Yang Chen, and Xin Wang are with the Shanghai Key Lab of Intelligent Information Processing, School of Computer Science, Fudan University, Shanghai 200438, China (e-mail: myzhou19@fudan.edu.cn; tcguo20@fudan.edu.cn; chenyang@fudan.edu.cn; xinw@fudan.edu.cn).

Yupeng Li is with the Department of Interactive Media, Hong Kong Baptist University, Hong Kong (e-mail: ivanypli@gmail.com).

Meng Niu is with Huawei Technologies Company Ltd., Beijing 100015, China (e-mail: niumeng3@huawei.com).

Pan Hui is with the Computational Media and Arts Thrust, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou 511453, China, also with the Division of Emerging Interdisciplinary Areas, The Hong Kong University of Science and Technology, Hong Kong, SAR, China, and also with the Department of Computer Science, University of Helsinki, 00560 Helsinki, Finland (e-mail: panhui@ust.hk).

Digital Object Identifier 10.1109/TNET.2024.3425227

that Polygon can more efficiently reschedule global resources compared with the commercial CDN schemes.

Our contributions are summarized as follows:

- We conduct a motivating case study illustrating that different applications prioritize different resource types when selecting CDN servers. Our goal is to handle delay-sensitive, bandwidth-sensitive, and CPU-sensitive CDN requests with an integrated solution.
- We propose Polygon, a QUIC-powered CDN server selection system that supports multiple resource demands. Polygon leverages QUIC's advantages to eliminate the extra delays and overhead introduced by the dispatcher.
- Real-world experiments and extensive simulations demonstrate Polygon's ability to reduce job completion time, improve resource utilization, and efficiently reschedule global resources with a moderate overhead.

A preliminary version of this paper has been published in [16]. The new contributions include design enhancements, implementation optimizations, comprehensive evaluations in real-world and simulation environments, and an in-depth discussion of Polygon's scalability and future research directions. In Section III, we improve Polygon's design on network resource measurements, server allocation algorithms, and resource weight vector calculations, making its operation more effective and efficient. Real-world deployment and extensive evaluations in various scenarios, presented in Section IV and Section V respectively, demonstrate the feasibility of deploying Polygon in production environments. Designed to be a resource-efficient CDN server selection system, Polygon can provide benefits including 1) less job completion time, 2) higher resource utilization, 3) fewer error requests, and 4) the capability to reschedule global resources dynamically.

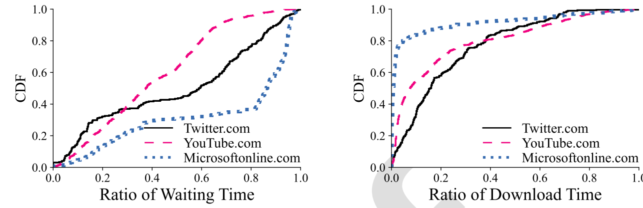
The following of this paper is organized as below. Section II introduces the insights that inspired Polygon. Section III describes the system design and implementation. A real-world evaluation is presented in Section IV, followed by extensive evaluations under various situations in Section V. Subsequently, Section VI discusses the scalability of Polygon and explores some future research. Section VII enumerates the related work. Finally, we conclude our work in Section VIII.

## II. Motivating Case Study

CDNs have evolved to support various content types, including web content [17], video streaming [18], and replica databases [19]. This section presents a case study revealing that CDN requests for different content types rely on distinct resource demands.

### A. Three CDN Request Patterns

We select three typical websites, Twitter.com, YouTube.com, and Microsoftonline.com, as case studies. These sites serve millions of users globally [20] and rely heavily on global CDN infrastructure [21], representing online microblogging, streaming media, and productivity tools, respectively.




(a) Ratio of waiting time (b) Ratio of download time

Fig. 1. Cumulative distribution functions of the ratios of waiting time and download time.

TABLE I
Resource Configurations in the Case Study

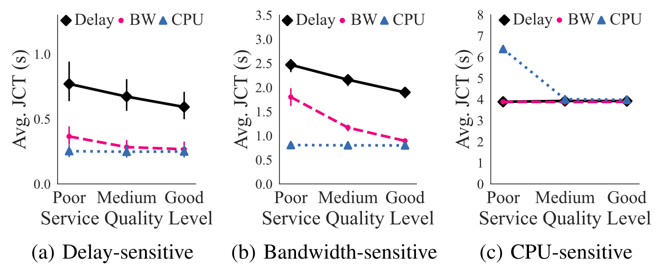
| Service Resource | Service Quality | Delay | Bandwidth | CPU |
|---|---|---|---|---|
| Delay | Poor | 131 ms | 248 Mbps | 1 vCPU |
| | Medium | 112 ms | 248 Mbps | 1 vCPU |
| | Good | 96 ms | 248 Mbps | 1 vCPU |
| Bandwidth | Poor | 34 ms | 3 Mbps | 1 vCPU |
| | Medium | 34 ms | 248 Mbps | 1 vCPU |
| | Good | 34 ms | 758 Mbps | 1 vCPU |
| CPU | Poor | 27 ms | 917 Mbps | 1 shared vCPU |
| | Medium | 27 ms | 917 Mbps | 1 vCPU |
| | Good | 27 ms | 917 Mbps | 4 vCPU |

We analyze the waiting time and download time [22], [23] of CDN requests on these three websites. According to Chrome's document [24], waiting time is defined as the duration from sending a request to receiving the first byte of the response, comprising one RTT and the server execution time. A longer waiting time indicates a longer server execution time given the same RTT delay. Download time is the duration spent receiving data, with a longer download time suggesting a slower network or larger data volume. These two parts constitute the majority of time to complete CDN requests [23]. We use Chrome-HAR[1] to capture the waiting time and download time, and treat each website entry as a CDN request. The capture process for these websites is conducted on the same machine under the same network conditions.

We calculate the ratios of waiting time and download time for each request. The Cumulative Distribution Functions (CDF) of these ratios for the three websites are shown in Fig. 1. Notably, over 60% of requests on Microsoftonline.com have a waiting time ratio exceeding 0.8, while Twitter.com and YouTube.com have fewer requests with such high waiting time ratios. In contrast, the download time ratios for requests on Microsoftonline.com are smaller than those on Twitter.com and YouTube.com.

These differences in waiting time and download time highlight that different services rely on distinct network resources. We introduce the concept of *resource sensitivity*: the degree to which request complete time changes due to variations in resource quality. Considering classic application scenarios, we categorize CDN requests into three sensitivity-related groups based on three common resource types: delay, bandwidth, and CPU capability. These resource types are widely recognized as representative of service resources [26].

[1] Chrome-HAR [25] is a file format that records session data of the browsing pages, including each entry's timestamps, load time, and size.




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

Fig. 2. JCT of the three CDN request types.

While other resources, such as network availability and storage ability, are also important, this study focuses on these three mainstream types as typical examples for simplicity.

- **Delay-sensitive** requests are sensitive to network delay, commonly found in activities such as *web browsing*, involving the retrieval of numerous small-sized contents from web pages.
- **Bandwidth-sensitive** requests are sensitive to available bandwidth, typically occur in *downloading* scenarios, including downloading large files or video streaming.
- **CPU-sensitive** requests are sensitive to CPU capability, frequently observed in *computing tasks* like database queries that demand high I/O and intensive computation.

## B. Verification on Resource Sensitivity

We conduct a case study on the Google Cloud Platform to verify the sensitivity of three CDN request types to different resources. The requests are emulated as follows. For delay-sensitive requests, we crawl the front pages of Alexa Top 500 Sites [20] and generate random visits to these pages. For bandwidth-sensitive requests, we use a 5MB video to generate a media CDN request. For CPU-sensitive requests, we execute 100 random queries on a database with one million entries. Each type generates 1,000 requests and is requested by a client running Ubuntu 18.04 LTS with one standard vCPU and 3.75 GB of memory. For resource setup of servers, we use "poor", "medium", and "good" to represent the servers' varying service quality levels in terms of delay, bandwidth, and CPU capability. Detailed configurations are listed in Table I.

We use job completion time (JCT) [27] as our metric. The results in Fig. 2 show significant differences in resource sensitivity of different request types. Delay-sensitive requests in Fig. 2(a) and CPU-sensitive requests in Fig. 2(c) exhibit reduced JCT as their dominant resource quality improves. Bandwidth-sensitive requests respond to changes in both bandwidth and delay, as shown in Fig. 2(b). Nevertheless, bandwidth still plays a dominant role. In comparison, JCT remains stable when irrelevant resource types change. Thus, to optimize CDN performance, server selection must consider multiple resource types rather than focusing on a single one.

# III. Design and Implementation

In this section, we present the design and implementation of Polygon. First, we show the overall workflow of our solution (Section III-A). Then, we list the design goals for implementing Polygon (Section III-B), where these

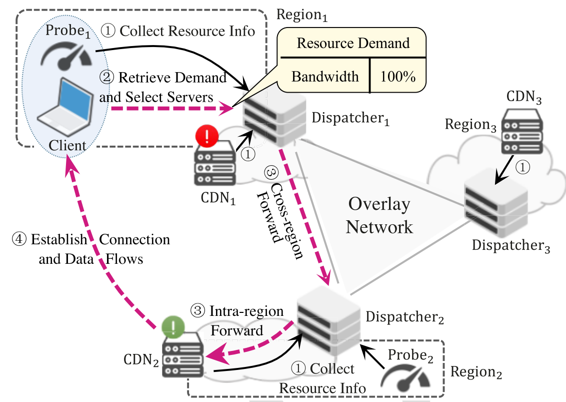



Fig. 3. Workflow of polygon for CDN server selection.

goals are embedded into the following three components: 1) scalable resource information collection (Section III-C), 2) adaptive resource demand design and allocation algorithm (Section III-E), and 3) low-latency connection and forwarding (Section III-F). Finally, we describe the detailed implementation (Section III-G).

## A. Workflow of Polygon

To realize multiple resource types perception, we propose Polygon, a QUIC-powered CDN server selection system. As depicted in Fig. 3, the workflow of Polygon is as follows:

*Step 1 (Collecting Resource Status Information):* Polygon allocates CDN servers based on requests' resource demands and current resource availability. Therefore, Polygon periodically collects resource information, including delay, bandwidth, and CPU capability, from widely deployed lightweight measurement probes (Section III-C).

*Step 2 (Retrieving Resource Demand and Selecting Suitable Servers):* Unlike previous solutions that directly send CDN requests to servers, the requests are first directed to an in-network dispatcher via anycast routing. Then, the dispatcher retrieves the request's resource demand set specified by the CDN provider or application developers (Section III-D) and selects suitable CDN servers using the Demand Restriction Allocation algorithm (Section III-E).

*Step 3 (Forwarding Request to Selected Server):* After selecting the appropriate CDN server, the dispatcher forwards the request to it. The server may be located in the same geographic region or in another. To reduce the delay caused by cross-region forwarding, Polygon establishes a fast-forwarding overlay network among dispatchers (Section III-F).

*Step 4 (Establishing Connection and Data Flows):* Upon receiving the request, the server sends a response with a migration signal to the client. Leveraging QUIC's connection migration function, the client seamlessly transfers the connection endpoint from the dispatcher to the server, avoiding the need to establish a new connection (Section III-F).

## B. Design Goals

In the above workflow, we involve three modules: 1) resource information collection, 2) resource demand design and allocation algorithm, and 3) connection and forwarding optimization. We systematically design these modules to facilitate Polygon with the following goals.

*Goal 1 (Efficient and Scalable Resource Status Monitoring):* The vast number of server-client pairs makes it impractical to measure the end-to-end links for all pairs within a reasonable timeframe. Additionally, due to the dynamics of the Internet, the resource status might be quite different between adjacent moments. Therefore, resource status monitoring must be efficient and scalable.

*Goal 2 (Adaptability to Diverse Usage Scenarios):* Polygon must be adaptive to diverse applications and varying expertise levels, supporting both automatic and customizable resource demand configurations. Moreover, the CDN allocation algorithm should effectively handle variable resource types and remain robust in various situations.

*Goal 3 (Minimize Delay in Connection and Forwarding):* Extra delay may occur due to the connection establishment for data flows and potential cross-regional forwarding. Minimizing such delay is crucial to maintain Polygon's advantages.

### C. Resource Status Collection

Various factors can influence CDN performance, including delay, bandwidth, network jitter, packet loss rate, and the capabilities of CPU, GPU, storage, and memory [26]. These factors are controlled by two main types of resources: network-related resources and hardware resources. For network-related resources, real-time monitoring of all end-to-end information is impractical. Therefore, we adopt a regional network aggregation strategy to keep monitoring costs modest. We deploy measurement probes to monitor network resources, with their results representing the network resource status of clients in the same region. Hardware resources are usually self-reporting, making their monitoring lightweight and scalable.

*1) Regional Network Aggregation:* Network resources are highly related to geographic location [28], [29]. Thus, we aggregate network resource information within a region by deploying probes with measurement functions.

**Region Definition.** Regions can be defined as geographically adjacent areas (e.g., provinces or cities) or network regions (e.g., autonomous systems). Boundaries are determined by noticeable differences in network conditions [29]. For instance, communications between endpoints in different cities will experience additional delays compared with communications within the same city.

**Measurement Probe.** Measurement probes are specialized devices for collecting and analyzing data on network performance and behavior, providing insights into Internet connectivity and issues identification. Platforms like RIPE Atlas, SamKnows, and BISmark [30] offer probe services. Probes are arranged near users at the city level in most regions. An analysis of the average RTT from Points of Presence (PoPs) to nearby clients showed that their RTT difference does not exceed 10 ms [31]. Additionally, we further conduct a case experiment to validate the effectiveness of probes in accurately representing client network conditions (Appendix).

*2) Available Capability Calculation:* A server's hardware includes processing capability, memory, and storage. Available capability is generally defined as the ratio of idle parts to the total capacity. Calculation methods vary for each resource. For example, available CPU capability is calculated as *idle rate* $\times$*number of CPU cores*$\times$ *CPU clock frequency*. Most hardware resources are equipped with well-developed and lightweight monitor tools that have a negligible impact on CPU overhead.

*3) Gathering Resource Information Into Dispatchers:* We introduce dispatchers to gather and manage the information about servers' resources. After each collection round, network resources and hardware resources information is delivered to each dispatcher, as depicted in Step 1 in Fig. 3. Dispatchers are not only responsible for periodically collecting resource information, but also for making server allocation decisions and forwarding requests to the selected CDN servers. Built on existing load balancing techniques such as Ananta [32], Maglev [33], Duet [34], dispatchers can handle millions of requests simultaneously. Dispatchers are strategically deployed in datacenters near major PoPs to forward requests to local and cross-regional CDN server clusters with minimal hops.

### D. Resource Demand Block

To represent request resource demands effectively and flexibly, we design a resource demand block that supports both pre-defined compositions and customizable configurations. We also propose a hybrid resource demand calculation that combines standard content profile categorization and resource sensitivity analysis to balance scalability and accuracy.

*1) Block Design:* We design a resource demand block to carry the resource demand information, structured into three parts: resource composition ID, resource type flag, and resource weight vector, as shown in Fig. 4. This design provides flexibility with two options: pre-defined demand compositions and customizable demand configurations.

Pre-defined resource compositions represent a set of typical resource demand configurations, each pre-configured with specific request flag values and resource weight vectors. Each composition is assigned a unique ID.

The pre-defined compositions only cover certain scenarios. Incorporating the resource type flag and weight vector allows for customized complex resource demand configurations. The flag field has 16 bits, with each bit corresponding to a resource type, which has been sufficient to cover the commonly used resource types. When a resource flag is set to True, it signifies sensitivity to that resource, with the detailed sensitivity value specified in the corresponding resource weight vector. Each resource has an 8-bit weight, representing the percentage of demand for that type relative to the total resource demand, ranging from 0 to 100. A larger weight indicates a greater demand for the corresponding resource type.

*2) Resource Weight Vector Calculation:* Assigning appropriate resource weight vectors to each CDN resource type is crucial for Polygon's effectiveness. However, this task is non-trivial, and manual allocation is not feasible. We propose a hybrid resource weight calculation that combines standard content profile categorization with resource sensitivity analysis to balance scalability and accuracy. Initially, CDN content

| Resource Demand Block | | | | | | |
|---|---|---|---|---|---|---|
| | Requested Resource Composition ID (16 bits) | | | | | |
| | Flag 1 (1 bit) | … | Flag 8 (1 bit) | Flag 9 (1 bit) | … | Flag 16 (1 bit) |
| | Res_1 Weight (8 bits) | | | Res_2 Weight (8 bits) | | |
| | … | | | … | | |
| | … | | | Res_16 Weight (8 bits) | | |

Fig. 4. Design of resource demand block.

could be classified into one of the standard profiles. If the content's behavior deviates from its assigned category's pattern, resource sensitivity analysis is employed to calculate a more precise resource weight vector.

**Standard Content Profiles and Benchmarks.** We establish standard content profiles that represent common CDN content types based on attributes like file type, size, and format. We have profiled small images, large videos, and resource retrieval. Each CDN content will be classified into one of these profiles based on its attributes, which enables quick and low-cost initial classification without explicit sensitivity analysis.

However, initial classification might not always be accurate. For instance, a video chunk might be classified as "media" due to its size. However, it should be reclassified as "quick fetch" since it is the beginning chunk of a video, where minimizing delay is more crucial. Automated webpage analysis tools like Lighthouse[2] can help identify misclassifications. These tools evaluate webpages by simulating loading activities and generate detailed reports with various metrics. Each profile is associated with a benchmark set that contains CDN performance under different resource conditions. Comparing Lighthouse's report with these benchmarks can verify the correctness of the initial classification. Significant deviations indicate misclassification and the necessity for adjustments.

**Resource Sensitivity Analysis.** When benchmark verification indicates that the CDN content does not align with the assigned category, resource sensitivity analysis will be conducted to calculate its resource weight vector. This vector, along with the CDN content attributes, will be recorded as a new profile.

Resource sensitivity analysis works by assessing performance differences across varying resource quality levels, reflecting the CDN content's sensitivity to a specific resource. We simulate environments with different resource quality levels, collect the corresponding load times, and compute the resource weight vectors. Specifically, each resource weight $w_j$ is calculated as follows: $w_j = \frac{t^j_{low} - t^j_{high}}{\sum_{j=1}^{n}(t^j_{low} - t^j_{high})}$, where $n$ is the number of resource types, $t^j_{low}$ is the load time under a low resource quality level, and $t^j_{high}$ is the load time under a high resource quality level.

*3) Default and Customizable Configuration:* Each hosted CDN content has a unique and stable resource weight vector calculated and stored by its provider. This vector can be initialized using the method described above when the CDN content is first declared to the provider. After receiving a request, the dispatcher retrieves the corresponding resource weight vector from the CDN provider by default, and then selects a CDN server using the algorithm described in Section III-E.

Polygon also supports customizable resource weight vectors on the client side, accommodating the diverse resource priorities of different users. This is achieved using the *QUIC Transport Parameters Extension*, which allows extra parameters to be transmitted during the handshake, enabling flexible configurations between clients and servers. With this function, authorized developers, who have permission to monitor and manage CDN content [35], can configure a customized resource weight vector to meet user requirements. However, customizable resource requirements may introduce risks of resource abuse and potential malicious behavior. Authentication mechanisms such as API keys and OAuth tokens [36], [37] can be used to verify the legitimacy of CDN requests. Note that if the next request's resource requirements differ from the previous one, a new connection will be launched. The new connections' cost can be eliminated using QUIC's 0-RTT connection resumption, as outlined in Section III-F.

### *E. Server Selection Algorithm*

In this section, we present our server selection algorithm called Demand Restriction Allocation (DRA). Its effectiveness lies in optimizing server allocation based on specified resource demands. The algorithm comprises two parts: server scoring and redundant forwarding. Server scoring ranks servers by assessing their maximum and currently available resources. Redundant forwarding enhances robustness in possible failed responses. The algorithm's pseudo-code is provided in Alg. 1. Note that this algorithm can be generalized to select a logical server, which may represent a compute cluster comprising multiple computational units, providing flexibility in allocation granularity according to scale and specific requirements.

*1) Server Scoring:* Two factors determine server $i$'s score: the capacity quota $Q_i$ and the available resources $A_i$ (line 2 to 4 of Alg. 1).

For a pending allocation request, the capacity quota $Q_i$ sets the upper limit of resources allocated to this request in server $i$. It is computed by proportionally distributing the total resource capacity among all connections. In line 2, we initially derive a unit of the capacity quota of resource $j$ by dividing the total capacity $r^{total}_{ij}$ by the sum of weights of $n$ connections and the pending allocation request. Then, this unit of capacity quota is multiplied by the weight $w_j$ to yield the capacity quota for resource $j$. Last, the capacity quotas for all resources are summed to get the overall capacity quota $Q_i$ for server $i$. The total capacity representation varies according to the resource type, but their values are all normalized from 0 to 1. The second factor, currently available resources $A_i$, is calculated by summing the availability of all resources in the server $i$ (line 3). This indicator is more instructive in situations where resources are not overloaded, enabling pending allocation requests to fully use remaining resources.

We set a threshold for cross-region forwarding operations (line 7). Cross-region forwarding might result in a performance

[2]https://developer.chrome.com/docs/lighthouse/overview

**Algorithm 1** Demand Restriction Allocation Algorithm for Server Selection With Redundant Forwarding

**Input:** Resource types $J = (j)_{j=1}^{16}$; Pending allocation request with resource demand vector $W = (w_j)_{j=1}^{16}$; Server list $S = (S_1, S_2, \ldots, S_m)$; Total capacity of server $i$ for resource type $j$ $r_{ij}^{total}$ and current available capacity $r_{ij}^{available}$.
**Output:** the optimal server $S_{1st}$ and the second_optimal server $S_{2nd}$.
*Initialization*: $S_{1st} \leftarrow$ NULL, $S_{2nd} \leftarrow$ NULL
1: **for** $S_i$**in**$S$ **do**
2: $Q_i = \sum_{j=1}^{16}(\frac{r_{ij}^{total}}{\sum_{k=1}^{n} w_j^k + w_j} * w_j)$
3: $A_i = \sum_{j=1}^{16} r_{ij}^{available}$
4: $S_i.score = Q_i + A_i$
5: **end for**
6: $candidate\_list \leftarrow$ sort($S$)
7: $candidate\_list \leftarrow$ optimize_with_threshold($candidate\_list$)
8: $S_{1st} \leftarrow$ get_optimal($candidate\_list$)
9: **if** $\frac{S_{1st}.score - S_{2nd}.score}{S_{1st}.score} < 10\%$ or $S_{2nd}$.RTT - $S_{1st}$.RTT $<$ 30 ms **then**
10: $S_{2nd} \leftarrow$ get_second_optimal($candidate\_list$)
11: **end if**
12: **return** $S_{1st}$, $S_{2nd}$

downgrade when the forwarding cost is higher than the gained benefit. Therefore, Polygon only selects those cross-region servers whose scores are higher than local CDN servers' scores by a certain degree.

*2) Redundant Forwarding:* To avoid possible response failures caused by potential sharp capacity degradation of the optimal server, we introduce a redundant forwarding mechanism (lines 8 to 10 of Alg. 1). Polygon selects both the optimal and the second optimal servers and forwards requests to both of them. This mechanism activates only when their score difference is below 10%, and the RTT difference is less than 30 ms. Accordingly, the client might receive two responses consecutively. The client only responds to the first received response and establishes a unicast connection with the corresponding server, while discarding other responses.

### *F. Request Forwarding*

The introduction of dispatchers inevitably brings extra delay, including the time for connection, forwarding, and client-server connection establishment. Polygon leverages QUIC to address this challenge. Compared with TCP + TLS 1.2, QUIC offers enhancements like lower latency handshakes (1-RTT and 0-RTT) and supports connection migration for seamless endpoint transfer.

*1) Quick Connection to Dispatchers:* Polygon uses anycast routing [9] to connect to the dispatcher with the shortest hops and optimizes handshake delay with QUIC's 1-RTT and 0-RTT mechanisms. In contrast to TCP, which requires 3 RTTs for transport and security handshakes, QUIC combines them into a 1-RTT handshake. The 0-RTT mechanism further optimizes delay. 0-RTT handshake in QUIC allows a client to resume a previous connection instantly by reusing a pre-shared key [38] retained before, eliminating the need for a full handshake. Frequent interactions between clients and dispatchers provide opportunities for the 0-RTT handshake.

Certainly, 0-RTT connections might be vulnerable to replay attacks [39], leading to unauthorized access. Thankfully, there are feasible solutions to secure 0-RTT connections [15], [39]. Additionally, 1-RTT connections have already effectively demonstrated QUIC's advantage in minimizing connection delays. The decision to employ the 0-RTT mechanism depends on the specific requirements and security considerations of the CDN provider.

*2) Fast-Forwarding via Overlay Network:* When the selected server and dispatcher are in the same datacenter or region, the dispatcher can directly forward requests through the CDN provider's intranet, where the forwarding delay is negligible [40]. However, forwarding requests to servers in other regions can result in higher delays [41]. To mitigate this, we construct an overlay network [42], [43] for fast-forwarding. An overlay network is a virtual network built on top of an existing physical network infrastructure, optimizing routing to bypass congested or slow links based on network topology and traffic patterns [44], [45].

The overlay network connects all dispatchers across regions. This means that the cross-regional forwarding follows the path "$dispatcher A \rightarrow dispatcher B \rightarrow server$". This hierarchical routing allows easy scaling of server capacity within the region without needing to check the entire topology. Major tech giants like Google [45] and Microsoft [46] have adopted this structure for inter-region data exchange.

*3) Mitigating Connection Delays Between Client and Server:* Finally, we leverage QUIC's connection migration mechanism to reduce connection delays between clients and selected servers. Unlike TCP-based CDN server selection, which requires establishing a new data flow connection after server assignment, QUIC supports seamlessly migrating connections from the dispatcher to the server, minimizing re-connection delays inherent in TCP-based systems. The detailed connection migration mechanism is described below.

QUIC specification includes a feature called *Server's Preferred Address* [14], allowing a server to accept connections on one IP address and transfer them to a preferred IP address shortly after the handshake, transitioning from anycast to more stable unicast [47]. This connection migration mechanism must adhere to the rule of ignoring packets received on addresses where migration has not started yet. To fulfil this requirement, we configure all dispatchers and servers to share the same anycast address, along with each server also having its unique unicast address. This setup guarantees that the server and dispatcher have the same IP address, meeting the conditions for initiating connection migration.

Upon receiving a handshake request forwarded by the dispatcher, the server initiates connection migration. In the handshake response to the client, the server includes the *preferred_address* parameter with its unicast address and sends this response via the anycast network interface. Since the server and dispatcher share the same anycast address, the client accepts the handshake response. The client parses the preferred_address and verifies the reachability of the preferred address. If the new preferred address is reachable, the client completes the connection establishment with the preferred address (i.e., the unicast address of the CDN server) and interrupts the old connection with the dispatcher.

This mechanism allows the client to run on the original connection for subsequent data transmission, eliminating the need for a new connection. With connection migration, Polygon efficiently handles requests, forwards requests, and establishes data transmission connections through one QUIC connection, significantly reducing delay.

### G. Implementation

Our prototype implementation consists of three key components: resource measurement, fast-forwarding overlay network, and deployment requirements for QUIC.

*1) Resource Measurement:* In our prototype, we monitor three typical resource types: network delay, bandwidth, and CPU capability. Network delay and bandwidth are monitored by probes. We obtain the network delay by Ping.[3] Available bandwidth is measured using the IGI/PTR [48]. CPU capability is reported with cpuacct.[4]

These resource collection intervals vary: delay is measured every 15 minutes, and bandwidth and CPU every 10 seconds. A 15-minute interval is sufficient to accurately characterize delay, as delay variations are generally below 10 ms and are insensitive to measurement intervals [48]. However, in cases of severe network congestion, significant delay changes can occur. Fixed measurement intervals might result in out-of-date resource information, affecting CDN server allocation accuracy. To address this, if bandwidth degrades by 30% and persists for 5 minutes, an extra delay measurement will be triggered to ensure adaptability and resilience. The cost of extra measurements is moderate due to the infrequent occurrence and low expense of delay measurements. Bandwidth and CPU experience frequent changes influenced by active transmission processes. To balance timeliness and accuracy, we set a 10-second measurement interval, which meets the duration requirements of most bandwidth testing services [49]. Our evaluations confirm the reasonableness of these interval settings, with Section IV demonstrating its effectiveness and Section V-E verifying the modest traffic overhead.

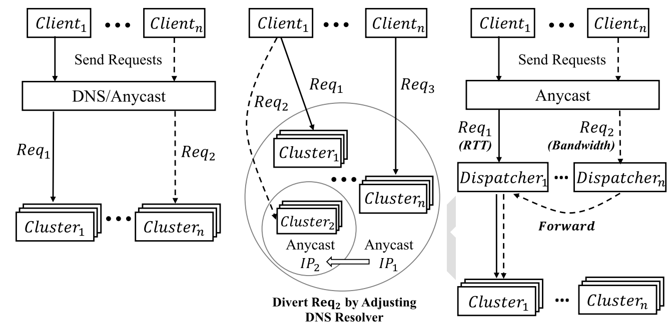
These values are specific to our prototype design and experimental environment. For real-world deployment, adjustments are necessary based on factors like deployment scale, real-time responsiveness requirements, and measurement overhead.

*2) Overlay Network:* To achieve quick forwarding among servers, we establish an overlay network that connects dispatchers using Generic Routing Encapsulation (GRE) tunnels. GRE tunnels are implemented with Open vSwitch [43], an open-source software switch supporting various tunneling protocols. To prevent multiple cross-region forwarding and network loops, we limit each request to be forwarded only once through the overlay network. If a dispatcher receives a request that has already been forwarded, it will directly forward the request to a local CDN server without exploring alternative servers in other regions.

*3) Deployment Requirements for QUIC:* Adopting Polygon requires a customized development on top of the QUIC protocol to implement the dispatcher. The dispatcher serves as

[3]https://linux.die.net/man/8/ping

[4]https://www.kernel.org/doc/Documentation/cgroup-v1/cpuacct.txt




(a) DNS/PureAnycast (b) FastRoute (c) Polygon

Fig. 5. Architectures of DNS, PureAnycast, FastRoute and Polygon.

a forwarding agent and is responsible for two main tasks: 1) parsing requests and determining their resource demand blocks and 2) forwarding the request to the appropriate CDN server. The first task requires the dispatcher to parse and modify request headers [50]. The second task involves encapsulating the request with GRE and forwarding it. As the dispatcher adheres to the QUIC specification and does not modify the packet structure, it is fully compatible with standard QUIC-based clients and servers. This customization is reasonable, given that the dispatcher is controlled by a CDN provider to improve user experience.

For client and server implementations, using mainstream browsers (e.g., Chrome,[5] Firefox[6]) and web server software (e.g., NGINX[7]) supporting QUIC connections is sufficient. The used Server's Preferred Address and connection migration functions are officially released in the QUIC specification [14] and implemented on the client side [50] and the server side [42]. Polygon can also work with other connection diversion techniques [8] to maintain compatibility when connection migration is not implemented or is disabled, albeit with some performance loss. On the basis of ngtcp2,[8] we develop the client, server, and dispatcher, along with the implemented Server's Preferred Address function [42]. The source code and documentation for our prototype are available at https://github.com/mengyingzhou/Polygon.

## IV. Evaluation in Real Network

This section presents the evaluation of Polygon. We begin by outlining the evaluation setup (Section IV-A). Next, we assess Polygon's performance in terms of JCT (Section IV-B) and resource utilization (Section IV-C). We then demonstrate Polygon's ability to reduce error requests under server overload conditions (Section IV-D) and show that this improvement is owing to appropriate cross-region request allocation (Section IV-E). Last, we display the necessity of redundant forwarding (Section IV-F).

### A. Experiment Setup

*1) Baselines:* We compare Polygon with three representative CDN server selection schemes: the widely used

[5]https://quiche.googlesource.com/quiche

[6]https://github.com/mozilla/neqo

[7]https://quic.nginx.org

[8]https://github.com/ngtcp2/ngtcp2

DNS-based CDN selection system [4], PureAnycast [51], and FastRoute [8]. Figure 5 illustrates the architectural differences between these schemes. The DNS-based and PureAnycast schemes share the same architecture shown in Figure 5(a).

- DNS-based scheme [4] allocates CDN servers by mapping the same domain name to different IP addresses using widely deployed regional DNS servers. This scheme is easy to operate but relies on large-scale DNS infrastructure and lacks awareness of server resource loads.
- PureAnycast [51] employs the raw anycast CDN selection approach, where multiple CDN servers broadcast the same IP address. Client requests are routed to the nearest CDN server with the fewest network hops according to Border Gateway Protocol (BGP). Despite its simplicity and cost-effectiveness, PureAnycast also lacks awareness of server resource loads.
- FastRoute [8] organizes CDN servers hierarchically to balance traffic under heavy loads. Servers close to clients form the outer layer, while backup servers constitute the inner layers. When the load on outer layer servers exceeds a threshold, FastRoute modifies DNS resolution to divert new requests to inner layer servers for load balancing. However, FastRoute may result in the underutilization of servers at the same layer, as the request redirection only occurs from the outer to inner layers, excluding within the same layer.

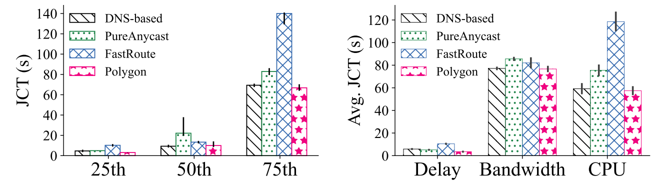
*2) Testbed Configuration:* We evaluate baselines and Polygon on the Google Cloud Platform. The platform restricts personal accounts to a total CPU quota of 10 [52]. As each VM requires at least one CPU, an account can only create up to 10 VMs simultaneously. Meanwhile, servers and dispatchers need to be on the same account for anycast functionality.[9] To mitigate this limitation, we use two accounts: one creates five servers and five dispatchers across five continents, and the other creates 10 clients (three in Asia, three in North America, two in Europe, one in Australia, and one in South America).

In addition to the server and client setup, each CDN scheme has its unique configurations: 1) Polygon requires one dispatcher per continent, implemented on Maglev [33], sharing the same anycast IP with servers[9]. 2) DNS-based scheme needs a DNS resolver, implemented with BIND.[10] 3) PureAnycast involves configuring all servers with the same anycast IP[9]. 4) FastRoute needs to build a virtual hierarchical architecture on servers. Following FastRoute's server placement design [8], we select servers close to most users (Asia, Europe, North America) for the outer layer and servers in regions with cheaper costs (Australia, South America) for the inner layer. FastRoute also requires a DNS resolver to adjust servers' IP addresses to realize request redirection.

*3) Evaluated Requests:* We construct the evaluated requests based on the three request types defined in Section II, with a ratio of 4:4:1 for delay-sensitive, bandwidth-sensitive, and CPU-sensitive requests, respectively. The ratio for CPU-sensitive requests is set lower due to limited server computing capacity. The three types of requests are single-resource sensitive, with only the weight of the sensitive resource set to 100 and the rest to 0. For example, bandwidth-sensitive requests have a resource vector of [0, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]. Each evaluation runs for about one hour, with five repetitions per scheme to mitigate network fluctuation effects.

[9]https://cloud.google.com/load-balancing

[10]https://www.isc.org/bind




(a) 25th percentile, 50th, and 75th percentiles of JCT (b) JCT of requests sensitive to different resource types

Fig. 6. JCT performance comparison.

We aim to evaluate Polygon's server allocation effectiveness under high concurrency. Due to quota constraints [52], we can only create 10 VMs as clients, each running ten processes simultaneously to simulate high concurrency. Although running multiple processes on one client may slightly reduce the realism, our evaluation on the Google Cloud Platform is still meaningful, providing practical results in a real network environment. Moreover, monitoring logs show that the client uses a maximum of 30% CPU and 5% memory, and the client bandwidth is set higher than the server's. Thus, running multiple processes on a client will not create additional bottlenecks. Additionally, to address this experimental limitation, we further evaluate Polygon under 105 clients in the follow-up simulations (Section V-B), achieving consistent performance with this setup using multiple processes per client.

*4) Metrics:* Three metrics are selected for performance evaluation:

- Job Completion Time (JCT) [27] is defined as the time taken to complete a CDN request. Specifically, JCT measures the duration from the start of the request to the end of the response data transmission.
- Cost per request (Cost / # Req.) [18] is defined as the average resource cost to complete each request. This metric measures the cost of bandwidth and CPU resources. For bandwidth, it is calculated as $\frac{\sum traffic}{\#\ of\ requests}$. For CPU, it is $\frac{\sum CPU\ usage}{\#\ of\ requests}$.
- Error ratio [53] is defined as the ratio of failed requests per second that the server does not execute successfully. Failures situations include unresponsive servers, interrupted connection, and processing timeouts.

## B. Less Job Completion Time

JCT is a key metric for evaluating CDN content fetching performance [54]. Fig. 6(a) depicts the JCT for the four methods at the 25th, 50th, and 75th percentiles. Polygon outperforms all baselines. Compared with the second-best method, Polygon reduces JCT by 37.5% at the 25th percentile, 5.8% at the 50th percentile, and 8.1% at the 75th percentile.

We further analyze what types of requests contribute to Polygon's performance superiority in Fig. 6(b). Polygon

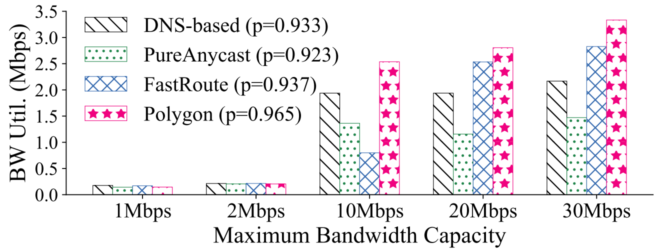



Fig. 7. Traffic utilization with varying bandwidth levels. $p$ is Pearson correlation coefficient between capacity and bandwidth utilization.

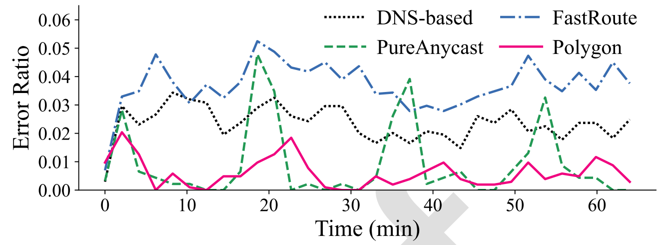



Fig. 8. Ratio of error requests over time.

TABLE II
REQUESTS THROUGHPUT AND AVERAGE RESOURCE COST FOR BANDWIDTH AND CPU

| 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** |

achieves the lowest average JCT values for all request types. Notably, for CPU-sensitive requests, Polygon shows a reduction of up to 57.7% compared with FastRoute. This is owing to Polygon redirecting 64.6% of these requests to less occupied regions, alleviating congestion on busy servers. A similar improvement is observed for bandwidth-sensitive requests.

Such redirection behavior also prevents performance degradation of connections in crowded regions, which is reflected in the reduced JCT for delay-sensitive requests. Unlike other requests, none of the delay-sensitive requests are forwarded to other regions (details discussed in Section IV-E). However, their JCT still decreases as Polygon prevents resource deprivation in local servers. By offloading downloading and computing tasks to other regions, delay-sensitive requests can use more resources to speed up completion. This indicates that request forwarding not only significantly reduces the JCT of forwarded requests but also enhances the performance of non-forwarded requests.

### C. Higher Resource Utilization

In addition to reducing JCT, Polygon improves overall server-side resource utilization and reduce request costs for service providers [12].

*1) Fully Leveraging Upgraded Bandwidth Resources:* We compare the bandwidth utilization of each scheme in detail. We manually limit the bandwidth capacity of servers to 1 Mbps, 2 Mbps, 10 Mbps, 20 Mbps, and 30 Mbps using Wonder Shaper.[11] In Fig. 7, we plot the bandwidth utilization of the four CDN schemes in five different levels of bandwidth.

At 1 Mbps and 2 Mbps, all schemes exhibit similar bandwidth utilization. However, as bandwidth capacity increases, the DNS-based scheme and PureAnycast do not show proportional increases in utilization, indicating they cannot effectively use upgraded resources. FastRoute shows a positive correlation between bandwidth capacity growth and utilization, but still lags behind Polygon.

To quantify the relationship between capacity and utilization, we calculate their Pearson correlation coefficient. A higher Pearson correlation coefficient $p$ indicates a stronger relationship between capacity and utilization. Polygon achieves the highest $p$ value of 0.965, followed by FastRoute at 0.937. The DNS-based scheme gets 0.933, and PureAnycast only gets 0.923. This indicates that considering resource demands helps to fully utilize upgraded resources.

*2) Reducing Request Costs:* Table II shows request throughput and average cost per request for bandwidth and CPU. Compared with PureAnycast, Polygon increases bandwidth-sensitive request throughput by 13% and reduces costs by 25% while also improving CPU-sensitive request throughput by 7% and reducing costs by 18%. PureAnycast and DNS-based schemes complete fewer requests and incur higher costs as they only allocate requests to local CDN servers. The local servers could be unavailable when flooded with requests.

In particular, FastRoute completes only 497 bandwidth-sensitive requests, just 23% of Polygon's total. Some requests are forwarded to unsuitable servers due to FastRoute's inability to consider request sensitivities. Another factor resulting in FastRoute's poor performance is its inflexible hierarchical load balancing, which restricts request redirection only from outer to inner layers, excluding within the same layer. This inflexibility, coupled with uneven global traffic distribution, leads to server overload in some regions while others remain idle, worsening resource utilization imbalance.

### D. Error Ratio

This subsection examines Polygon's ability to handle error requests when servers are overloaded. An error request occurs when the server fails to execute successfully due to unresponsiveness, interrupted connections, or processing timeouts. We plot the request error ratio over time in Fig. 8. Initially, each scheme exhibits a reasonable error ratio, but the DNS-based scheme and PureAnycast experience a rapid error ratio increase over time. These schemes allocate CDN servers solely based on delay, sending requests to the local server regardless of its resource load.

The error ratio peaks almost every 20 minutes due to accumulated CPU-sensitive requests overloading CPU resources. High CPU load can cause servers to halt, negatively impacting the user experience and bringing recovery costs. FastRoute suffers the most from such server halting problems,

[11]https://github.com/magnific0/wondershaper

TABLE III
CROSS-REGION FORWARDING RATIO OF FASTROUTE (F) AND POLYGON (P)

| Resource | Delay | | BW | | CPU | |
|---|---|---|---|---|---|---|
| Method | F | P | F | P | F | P |
| Ratio | 0.654 | 0 | 0.116 | 0.474 | 0.723 | 0.646 |

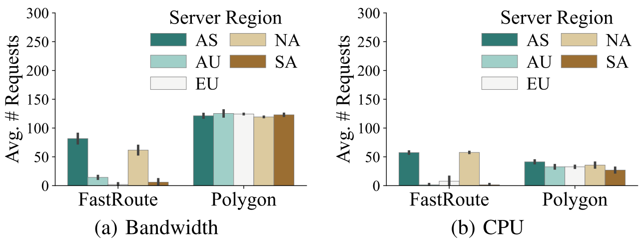



Fig. 9. Requests completed per region with FastRoute vs. Polygon.

displaying the highest error ratio. Due to FastRoute's inability to consider multiple resource types, redirecting requests to backup servers can still lead to single point congestion, even if backup servers temporarily alleviate some pressure.

In comparison, Polygon achieves the lowest error request ratio throughout the experiment. When a server becomes crowded in one resource type, Polygon forwards corresponding requests to unoccupied servers in other regions, balancing resource loads and reducing congestion. Notably, Polygon's error ratio peaks occur later than others, suggesting its effectiveness in delaying resource congestion.

### E. Forwarding Behavior

FastRoute is the most relevant baseline to Polygon, as it also forwards requests to other regions. By comparing their forwarding behavior, we identify two key properties that make Polygon's forwarding superior: 1) forwarding appropriate requests only when necessary; 2) fair allocation of requests across regions.

*1) Forwarding Appropriate Requests:* We examine the ratio of cross-region forwarded requests with different resource sensitivities in Table III. As expected, Polygon's forwarding ratio for delay-sensitive requests is zero. Generally, nearby CDN servers are optimal for handling delay-sensitive requests due to their shorter delay. In contrast, FastRoute forwards 65.4% of delay-sensitive requests to the inner layer. This results in many delay-sensitive requests being incorrectly redirected to farther servers just because their less-relevant resources are overloaded. This forwarding property further explains the poor performance of delay-sensitive requests using FastRoute.

*2) Fair Allocation of Requests:* In Table III, we also observe significant differences in forwarding ratios for bandwidth-sensitive and CPU-sensitive requests between Polygon and FastRoute. Therefore, in Fig. 9, we further compare the number of completed requests in each region between FastRoute and Polygon. It is evident that Polygon allocates bandwidth-sensitive and CPU-sensitive requests more evenly by precisely understanding resource requirements. Moreover, this fair allocation makes Polygon less susceptible to the "herding effect" problem.

In contrast, FastRoute shows obviously uneven allocation due to its constraint of redirecting requests only from outer to inner layers, excluding within the same layer. FastRoute's architecture includes an outer layer with servers in Asia, Europe, and North America, and an inner layer with servers in other regions. Figure 9 shows that servers in Asia and North America handle significantly more requests than those in Europe, despite all belonging to the outer layer, primarily due to more users in Asia and North America. However, FastRoute cannot redirect requests within the outer layer, leading to a continuously imbalanced request pattern and worsening its performance.

### F. Redundant Forwarding

To enhance Polygon's robustness and avoid potential response failures, we introduce a redundant forwarding mechanism that triggers under specific conditions. The client prioritizes the first response and discards the latter one. Our experiments show that an average of 10% of requests trigger redundant forwarding, with 2% of connections established by the second optimal server. Enabling this mechanism improves median JCT from 22.55 s to 20.59 s and reduces the overall error ratio from 7.1% to 6.4%. This demonstrates that redundant forwarding reduces errors and enhances responsiveness.

## V. SIMULATION

This section explores Polygon's performance in various simulations. We introduce the simulation setup (Section V-A), present JCT performance and throughput at scale (Section V-B), discuss the impact of resource arrangement and Polygon's capability to reschedule resources (Section V-C), study the benefits of cross-region requests under different network conditions (Section V-D), and analyze Polygon's overhead from a scalability perspective (Section V-E).

### A. Simulation Setup

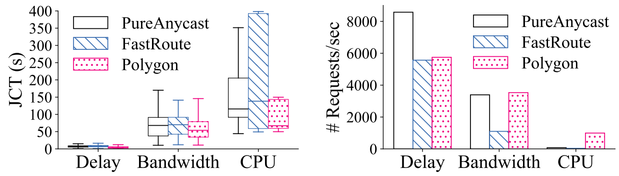
We create simulation environments using Mininet,[12] a widely used network emulator creating a realistic virtual network with running real kernels, switches, and application codes. The host running Mininet is configured with 64 CPU cores and 187 GB of memory. To mimic real-world network conditions, we collect network information between each pair of regions and zones[13] on the Google Cloud Platform. We collect data over a week and calculate the median value to represent each pair's network condition.

The deployment setup is listed in Table IV. There are 105 clients and 15 servers, with one dispatcher per region. Here, we largely increase the number of test machines using the Mininet emulator to address the scalability limitations of real-world experiments caused by quota restrictions in Section IV. The baselines compared in this section are PureAnycast and FastRoute.

[12]http://mininet.org
[13]https://cloud.google.com/compute/docs/regions-zones

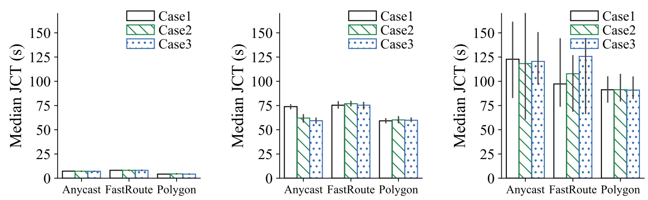
TABLE IV
SIMULATION SETUP

| Region | # Clients | # Servers |
|---|---|---|
| Asia | 28 | 4 |
| Australia | 7 | 1 |
| Canada | 14 | 2 |
| Europe | 21 | 3 |
| South America | 7 | 1 |
| United States | 28 | 4 |




(a) Boxplot of JCT (b) Request Throughput

Fig. 10. JCT and request throughput comparisons of PureAnycast, FastRoute, and Polygon at a large scale.




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

Fig. 11. JCT comparisons of PureAnycast, FastRoute, and Polygon under different resource arrangement cases. Case 1: improving crowded-request's servers. Case 2: improving uncrowded-request's servers. Case 3: improving one random server in each region.

### B. Large-Scale Evaluation

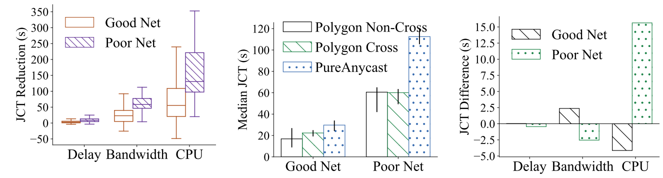
We evaluate Polygon's performance at scale using the above setup, comparing its JCT and throughput with PureAnycast and FastRoute. Results in Fig. 10 show Polygon outperforms the other two schemes in JCT for all three request types while maintaining comparable throughput. Notably, for CPU-sensitive requests, Polygon achieves significant improvement over PureAnycast, with reducing JCT by 42.1% and increasing throughput by 13x.

An unexpected finding observed in Fig. 10(b) is that for delay-sensitive requests, PureAnycast shows higher throughput than Polygon and FastRoute. Further analysis reveals that most delay-sensitive requests completed in the PureAnycast scheme come from non-crowded regions (Australia and South America). These regions have fewer bandwidth-sensitive and CPU-sensitive requests, meaning servers are not fully loaded and have sufficient local capacity to handle more delay-sensitive requests.

### C. Resource Arrangement

*1) Resource Arrangement Setup:* Resources are not evenly distributed in most real-world scenarios, requiring CDN service providers to adjust resources based on request volume manually. Proper resource arrangement is crucial for optimizing CDN response speed. We design three resource arrangement cases based on the deployment setup as in Section V-A. 1) Case 1: arranging more powerful servers in crowded regions (Asia and the United States). 2) Case 2: arranging more powerful servers in the less crowded regions (Australia and South America). 3) Case 3: one random server in each region is upgraded, doubling/tripling its bandwidth and CPU capacity, with the total resource capacity kept the same across all cases.

*2) Polygon's Capability for Resource Rescheduling:* This experiment underlines the stability of JCT with Polygon, highlighting its capability for resource rescheduling under different resource arrangements. As shown in Fig. 11, Polygon maintains consistent JCT performance across different resource arrangements (maximum standard deviation is 26.0). By contrast, Anycast and FastRoute exhibit larger variations (maximum standard deviations are 81.4 and 65.4, respectively). For these two schemes, resource arrangement is a fragile factor for performance since human-manipulated configuration might not be optimal for every scenario. By perceiving the requests' resource sensitivity, Polygon addresses this drawback and adaptively reschedules global resources, mitigating the performance impact of different arrangements.




(a) JCT reduction of Polygon compared with PureAnycast under different network conditions (b) JCT of Polygon's non-cross-region, Polygon's cross-region, and PureAnycast requests (c) JCT difference between cross-region and non-cross-region requests

Fig. 12. JCT optimization from cross-region forwarding under good and poor network conditions.

### D. Different Network Environments

*1) More JCT Reduction Benefit Under Poor Network Conditions:* Here, we examine how network conditions impact Polygon's performance by creating two network environments: a good network condition (average bandwidth of 2.72 Mbps and average RTT of 42.5 ms) and a poor network condition (average bandwidth of 0.77 Mbps and average RTT of 144.6 ms).

We view network conditions with an RTT of less than 100 ms as "good", and conversely as "poor". We use this criterion to divide the real network condition data collected in Section V-A into two categories. The deployment setup remains consistent with that described in Section V-A for both network cases. The good network case is created by randomly configuring network conditions between machines with the good category, while the poor network case is constructed similarly but with using the poor category.

Fig. 12(a) shows that Polygon achieves more JCT reduction under poor network conditions, demonstrating Polygon is more helpful in severe network conditions. To investigate its reason, we analyze the JCT of Polygon's non-cross-region requests, Polygon's cross-region requests, and PureAnycast requests in Fig. 12(b). It can be found that under poor network conditions, cross-region requests can achieve JCT comparable to non-cross-region requests. This is owing to Polygon's adaptive forwarding strategy, which selects better options during request congestion. In contrast, PureAnycast experiences more severe congestion and poorer performance due to its lack of flexibility. While under good network conditions, request congestion is less severe, making Polygon's cross-region requests benefit less obvious. However, Polygon's resource scheduling still improves non-cross-region request performance by forwarding a portion of requests to unoccupied servers.

*2) Impact of Network Conditions on Requests Sensitive to Different Resource Types:* Fig. 12(c) illustrates the JCT difference between cross-region and non-cross-region requests for each request type. The y-axis represents the median JCT of non-cross-region requests minus the median JCT of cross-region requests, i.e., $JCT_{non\_cross} - JCT_{cross}$. Bars above the x-axis indicate that cross-region requests perform better than non-cross-region requests. We find that different request types are affected differently under these two network conditions. For delay-sensitive and bandwidth-sensitive requests, cross-region forwarding under poor network conditions may degrade performance due to additional delays. However, CPU-sensitive requests, which are less affected by network conditions, significantly benefit from being forwarded to unoccupied servers.

### E. Overhead

In this subsection, we examine the overhead of Polygon. Six metrics are used to assess the overhead from the perspectives of client scalability and server scalability.

- Client scalability: 1) CPU usage of dispatchers, 2) forwarding traffic volume, and 3) forwarding delay. These metrics quantify the overhead of dispatchers in relation to the number of clients.
- Server scalability: 4) measurement traffic volume on servers, 5) CPU usage of servers, and 6) query and ranking delay. These metrics reflect the overhead of resource status measurement on the CDN servers and dispatchers, which are related to the number of servers.

The overhead results of scalability experiments are shown in Table V. For client scalability, the CPU usage on a dispatcher is only 60.74% in the case of 2,000 clients, not yet reaching full load. The traffic caused by request forwarding is only 0.722 Mbps, which is a negligible cost for global CDN deployment. The forwarding delay is at most 2.551 ms, an imperceptible delay for requests. For server scalability, the measurement traffic volume on servers is 6.438 Kbps, using only 5.31% of CPU capacity in the case of 15,000 servers. Query and ranking delay is up to only 225.22 ms. Overall, these results demonstrate that Polygon can provide quick-responsive CDN services and handle high-request concurrency without excessive overhead.

TABLE V
OVERHEAD OF POLYGON FROM THE PERSPECTIVE OF CPU USAGE, TRAFFIC, AND DELAY

| **# Clients** | **100** | **300** | **500** | **1000** | **2000** |
|---|---|---|---|---|---|
| CPU Usage of Dispatchers (%) | 7.50 | 17.61 | 34.59 | 54.71 | 60.74 |
| Forwarding Traffic (Mbps) | 0.023 | 0.091 | 0.183 | 0.366 | 0.722 |
| Forwarding Delay (ms) | 0.019 | 0.060 | 0.156 | 1.109 | 2.551 |
| **# Servers** | **100** | **1000** | **5000** | **10000** | **15000** |
| Measurement Traffic (Kbps) | 0.232 | 1.305 | 4.193 | 5.571 | 6.438 |
| CPU Usage of Servers (%) | 1.830 | 2.703 | 4.163 | 5.182 | 5.316 |
| Query and Ranking Delay (ms) | 1.80 | 8.18 | 33.70 | 100.16 | 225.22 |

## VI. DISCUSSION

This section discusses the scalability considerations of Polygon (Section VI-A) and future research to enhance its performance (Section VI-B).

### A. Scalability Considerations

In this subsection, we explore Polygon's scalability, focusing on placement strategy and dispatcher density and cost.

**Scalability for placement strategy:** Currently, we follow a strategy aligned with commercial datacenters, placing dispatchers near major PoPs. This strategy is both cost-effective and efficient for covering a wide range of regions and users, as the locations and densities of these commercial datacenters have been optimized and validated in practice [55]. Additionally, Polygon could be implemented on commodity servers without special hardware. This allows us to make use of the idle edge servers as dispatchers, thus reducing deployment costs. Our overhead experiments (Section V-E) show that commodity edge servers are sufficient for running dispatcher programs. Moreover, the evolution and expansion of edge servers over the years have ensured broad coverage, offering feasibility to meet various placement densities.

**Scalability for deployment density:** A modest number of dispatchers per region have already effectively managed a high volume of requests. Our overhead results (Section V-E) show that five dispatchers can handle 2,000 concurrent requests and monitor 15,000 servers' resource information. Even with minimal configurations (1 vCPU and 3.75 GB memory), dispatchers perform well. This is due to two optimizations: 1) Dispatchers only process the header part during connection setup, without parsing data packets. 2) Dispatchers handle incoming requests but not outbound CDN traffic. Connections for CDN data flows are established directly between servers and clients, bypassing the dispatcher. Consequently, a few strategically placed dispatchers are sufficient for Polygon to handle global requests efficiently.

### B. Future Research

Future enhancements for Polygon in production environments include:

**Exploring the deployment strategy of dispatchers.** Optimizing the placement strategy of dispatchers is a key direction for future research. We plan to adopt and refine existing solutions that consider geographical location [56], deployment cost [57], [58], and resource utilization [59] to further enhance Polygon's performance and efficiency.

**Implementing Polygon over other transport layer protocols.** Extending Polygon to support protocols other than QUIC, such as TCP, would broaden its applicability. Although this may reduce some of the low-latency benefits of QUIC adoption, it would provide compatibility with numerous existing TCP-based services and applications. We believe that optimization efforts from the TCP research community could offer alternatives to achieve performance comparable to QUIC.

**Evaluating and refining Polygon in browser environments.** Our experiments demonstrate Polygon's capability to optimize CDN performance at the request level. We plan to test Polygon in more complicated browsing scenarios, with considering webpage structure and browser loading behavior. Future evaluations will use page-level metrics such as Speed Index and First Content Paint to further validate and optimize Polygon's performance in real-world browsing environments.

## VII. Related Work

### A. Anycast-Based CDN

Anycast is a fundamental technology in modern CDNs, aligning well with the CDN concept of fetching Internet content from nearby servers. Flavel et al. [8] proposed FastRoute, a hierarchical anycast-based approach that directs users to the nearest service replicas and has the ability to balance request load. This approach was successfully deployed on the Microsoft Bing search engine [10]. Despite its good performance in server selection, FastRoute faced a control loss problem, directing about 20% of requests to suboptimal endpoints [60]. To address this, Alzoubi et al. [5], [6] developed a load-aware anycast CDN routing using server and network load feedback for better redirection control. Fu et al. [7] introduced T-SAC, employing a 1-bit non-redirection flag for fine-grained traffic control. Additionally, Lai and Fu [47] suggested converting a CDN server's anycast connection to unicast connection via their MIMA middleware to prevent connection interruptions.

### B. Load Balancing

Load balancing is a crucial component in Internet-scale distributed systems. Ananta [32], introduced by Patel et al. in 2013, and Maglev [33], proposed by Eisenbud et al. in 2016, are well-known load balancers deployed on the large-scale networks infrastructure of Microsoft and Google, respectively. Apart from balancing traffic volumes, other considerations have driven research in this area. Mathew et al. [12] took energy optimization as the primary principle and designed an energy-aware algorithm to reduce consumption. Zhang et al. [61] addressed load balancing for scenarios under uncertainties, improving performance when switches occasionally failed. Miao et al. [62] utilized switching ASICs to build faster load balancers, which were capable of handling 10 million connections simultaneously. Gandhi et al. [34] embedded the load balancing function into hardware switches, achieving 10x in capacity and 1/10 in delay than software-based solutions.

## VIII. Conclusion

This paper proposes Polygon, a CDN server selection system that perceives multiple resource demands based on QUIC protocol and anycast routing. Equipped with well-designed dispatchers and measurement probes, Polygon identifies suitable CDN servers for requests based on resource requirements and server availability. Leveraging QUIC's 0-RTT and connection migration features, Polygon establishes fast connections and expedites client-server pairing. Additionally, Polygon minimizes request forwarding delays across regions through a fast-forwarding overlay among dispatchers. Evaluations in real-world environments and simulation testbeds demonstrate Polygon's capability to enhance QoE, optimize resource utilization, and dynamically reschedule resources.

## Appendix
## Effectiveness of Probe Representations

We conduct a case study to verify the effectiveness of using probes to represent the network conditions of a region. We deploy two clients in Shanghai, China. One is connected to the Internet via a residential wired network, and the other via a cellular network. A probe node is placed in a datacenter of Alibaba Cloud in the same city. Two servers are located in Wisconsin and Utah, each with a maximum network capacity of 100Mbps.

To assess the similarity between the measurements obtained by the clients and the probe, both clients and the probe simultaneously measure available bandwidth to the servers using the IGI/PTR [48], a lightweight bandwidth measurement tool. Tests are conducted three times, with each round lasting one hour and spaced eight hours intervals. We use the Spearman Correlation Coefficient [63] as the similarity metric, which ranges from $-1$ to $+1$, with values closer to $+1$ indicating higher positive correlation. The correlation coefficient between the wired client and the probe is 0.845, and that between the cellular client and the probe is 0.805. These results align with prior research [30], [64], confirming that network conditions measured by nearby probes can accurately represent those experienced by clients.

## References


[1] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *Proc. USENIX NSDI*, 2017, pp. 67–82.

[2] K. Psychas and J. Ghaderi, "Randomized algorithms for scheduling multi-resource jobs in the cloud," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2202–2215, Oct. 2018.

[3] A. Munir, T. He, R. Raghavendra, F. Le, and A. X. Liu, "Network scheduling and compute resource aware task placement in datacenters," *IEEE/ACM Trans. Netw.*, vol. 28, no. 6, pp. 2435–2448, Dec. 2020.

[4] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, 2010.

[5] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van der Merwe, "Anycast CDNs revisited," in *Proc. WWW*, 2008, pp. 277–286.

[6] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van Der Merwe, "A practical architecture for an anycast CDN," *ACM Trans. Web*, vol. 5, no. 4, pp. 1–29, Oct. 2011.

[7] Q. Fu et al., "Taming the wild: A scalable anycast-based CDN architecture (T-SAC)," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2757–2774, Dec. 2018.

[8] A. Flavel et al., "FastRoute: A scalable load-aware anycast routing architecture for modern CDNs," in *Proc. USENIX NSDI*, 2015, pp. 381–394.

[9] C. Partridge, T. Mendez, and W. Milliken. *Host Anycasting Service*. Accessed: Jun. 1, 2024. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1546.html

[10] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the performance of an anycast CDN," in *Proc. ACM IMC*, 2015, pp. 531–537.

[11] H. Shen and L. Chen, "Resource demand misalignment: An important factor to consider for reducing resource over-provisioning in cloud datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1207–1221, Jun. 2018.

[12] V. Mathew, R. K. Sitaraman, and P. Shenoy, "Energy-aware load balancing in content delivery networks," in *Proc. IEEE INFOCOM*, 2012, pp. 954–962.

[13] A. Langley et al., "The QUIC transport protocol: Design and Internet-scale deployment," in *Proc. ACM SIGCOMM*, 2017, pp. 183–196.

[14] J. Iyengar and M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Accessed: Jun. 1, 2024. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9000.html

[15] F. Günther, B. Hale, T. Jager, and S. Lauer, "0-RTT key exchange with full forward secrecy," in *Proc. Eurocrypt*, 2017, pp. 519–548.

[16] M. Zhou, T. Guo, Y. Chen, J. Wan, and X. Wang, "Polygon: A QUIC-based CDN server selection system supporting multiple resource demands," in *Proc. ACM/IFIP Middleware*, 2021, pp. 16–22.

[17] L. Zhang, F. Zhou, A. Mislove, and R. Sundaram, "Maygh: Building a CDN from client web browsers," in *Proc. ACM EuroSys*, 2013, pp. 281–294.

[18] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, "Practical, real-time centralized control for CDN-based live video delivery," in *Proc. ACM SIGCOMM*, 2015, pp. 311–324.

[19] N. Khaitiyakun and T. Sanguankotchakorn, "An analysis of data dissemination on VANET by using content delivery network (CDN) technique," in *Proc. ACM AINTEC*, 2014, pp. 37–42.

[20] Alexa Internet. (2021). *The Top 500 Sites on the Web*. Accessed: May 1, 2022. [Online]. Available: https://www.alexa.com/topsites

[21] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting video server selection strategies in the Youtube CDN," in *Proc. IEEE ICDCS*, 2011, pp. 248–257.

[22] Y.-M. Chiu and D. Y. Eun, "Minimizing file download time in stochastic peer-to-peer networks," *IEEE/ACM Trans. Netw.*, vol. 16, no. 2, pp. 253–266, Apr. 2008.

[23] K. Hayashi, R. Ooka, T. Miyoshi, and T. Yamazaki, "P2PTV traffic classification and its characteristic analysis using machine learning," in *Proc. IEEE APNOMS*, 2019, pp. 1–6.

[24] K. Basques. (2021). *Timing Breakdown Phases Explained*. Accessed: Jun. 1, 2024. [Online]. Available: https://developer.chrome.com/docs/devtools/network/reference/#timing-explanation

[25] A. Cardaci. (2017). *Chrome HAR Capturer*. Accessed: Jun. 1, 2024. [Online]. Available: https://github.com/cyrus-and/chrome-har-capturer

[26] B. Zolfaghari et al., "Content delivery networks: State of the art, trends, and future roadmap," *ACM Comput. Surveys*, vol. 53, no. 2, pp. 1–34, Mar. 2021.

[27] F. Lai et al., "Sol: Fast distributed computation over slow networks," in *Proc. USENIX NSDI*, 2020, pp. 273–288.

[28] C. Bovy, H. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal, and P. Van Mieghem, "Analysis of end-to-end delay measurements in Internet," in *Proc. PAM*, 2002, pp. 1–8.

[29] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proc. USENIX NSDI*, 2015, pp. 323–336.

[30] V. Bajpai and J. Schönwälder, "A survey on Internet performance measurement platforms and related standardization efforts," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1313–1341, 3rd Quart., 2015.

[31] M. Williams. (2020). *Website Latency With and Without a Content Delivery Network*. Accessed: Jun. 1, 2024. [Online]. Available: https://www.keycdn.com/blog/website-latency

[32] P. Patel et al., "Ananta: Cloud scale load balancing," in *Proc. ACM SIGCOMM*, 2013, pp. 207–218.

[33] D. E. Eisenbud et al., "Maglev: A fast and reliable software network load balancer," in *Proc. USENIX NSDI*, 2016, pp. 523–535.

[34] R. Gandhi et al., "Duet: Cloud scale load balancing with hardware and software," in *Proc. ACM SIGCOMM*, 2014, pp. 27–38.

[35] U. Naseer and T. A. Benson, "Configanator: A data-driven approach to improving CDN performance," in *Proc. USENIX NSDI*, 2022, pp. 1135–1158.

[36] D. Fett, B. Campbell, J. Bradley, T. Lodderstedt, M. B. Jones, and D. Waite. *OAuth 2.0 Demonstrating Proof of Possession (DPoP)*. RFC 9449. Accessed: Jun. 1, 2024. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9449.html

[37] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS meets CDN: A case of authentication in delegated service," in *Proc. IEEE S&P*, 2014, pp. 67–82.

[38] M. Thomson and S. Turner. *Using TLS To Secure QUIC*. RFC 9001. Accessed: Jun. 1, 2024. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9001.html

[39] M. Fischlin and F. Günther, "Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates," in *Proc. IEEE EuroS&P*, 2017, pp. 60–75.

[40] W. Li, D. Guo, K. Li, H. Qi, and J. Zhang, "IDaaS: Inter-datacenter network as a service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1515–1529, Jul. 2018.

[41] G. Zeng et al., "Congestion control for cross-datacenter networks," in *Proc. IEEE ICNP*, 2019, pp. 1–12.

[42] X. Li et al., "Artemis: A latency-oriented naming and routing system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4874–4890, Dec. 2022.

[43] B. Pfaff et al., "The design and implementation of open vSwitch," in *Proc. USENIX NSDI*, 2015, pp. 117–130.

[44] S. Bhatia et al., "Trellis: A platform for building flexible, fast virtual networks on commodity hardware," in *Proc. ACM CoNEXT*, 2008, pp. 1–6.

[45] C.-Y. Hong et al., "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN," in *Proc. ACM SIGCOMM*, 2018, pp. 74–87.

[46] F. Abuzaid, S. Kandula, B. Arzani, I. Menache, M. Zaharia, and P. Bailis, "Contracting wide-area network topologies to solve flow problems quickly," in *Proc. USENIX NSDI*, 2021, pp. 175–200.

[47] J. Lai and Q. Fu, "Man-In-the-Middle anycast (MIMA): CDN user-server assignment becomes flexible," in *Proc. IEEE LCN*, 2016, pp. 451–459.

[48] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 6, pp. 879–894, Aug. 2003.

[49] X. Yang et al., "Fast and light bandwidth testing for Internet users," in *Proc. USENIX NSDI*, 2021, pp. 1011–1026.

[50] *Parsing QUIC Client Hellos*. Accessed: Jun. 1, 2024. [Online]. Available: https://www.chromium.org/quic/parse-client-hello

[51] A. Barbir, B. Cain, R. Nair, and O. Spatscheck. *Known Content Network (CN) Request-Routing Mechanisms*. RFC 3568. Accessed: Jun. 1, 2024. [Online]. Available: https://www.rfc-editor.org/info/rfc3568

[52] Google LLC. (2022). *Quotas & Limits on Google Cloud Platform*. Accessed: Jun. 1, 2024. [Online]. Available: https://cloud.google.com/compute/resource-usage

[53] T. Yang et al., "Elastic Sketch: Adaptive and fast network-wide measurements," in *Proc. ACM SIGCOMM*, 2018, pp. 561–575.

[54] J. Zhang et al., "WiseTrans: Adaptive transport protocol selection for mobile web service," in *Proc. ACM WWW*, 2021, pp. 284–294.

[55] L-Google Co. *How Does Google Select a Data Center Location*. Accessed: Jun. 1, 2024. [Online]. Available: https://www.google.com/about/datacenters/discover

[56] B. Yu and J. Pan, "Location-aware associated data placement for geo-distributed data-intensive applications," in *Proc. IEEE INFOCOM*, 2015, pp. 603–611.

[57] J. Mudigonda, P. Yalagandula, and J. C. Mogul, "Taming the flying cable monster: A topology design and optimization framework for data-center networks," in *Proc. USENIX ATC*, 2011, pp. 1–33.

[58] M. Zhang, R. N. Mysore, S. Supittayapornpong, and R. Govindan, "Understanding lifecycle management complexity of datacenter topologies," in *Proc. USENIX NSDI*, 2019, pp. 235–254.

[59] X. Xu et al., "An IoT-oriented data placement method with privacy preservation in cloud environment," *J. Netw. Comput. Appl.*, vol. 124, pp. 148–157, Dec. 2018.

[60] Z. Li, D. Levin, N. Spring, and B. Bhattacharjee, "Internet anycast: Performance, problems, & potential," in *Proc. ACM SIGCOMM*, 2018, pp. 59–73.

[61] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. ACM SIGCOMM*, 2017, pp. 253–266.

[62] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. ACM SIGCOMM*, 2017, pp. 15–28.

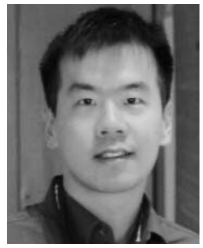
[63] C. Spearman, "The proof and measurement of association between two things," *Amer. J. Psychol.*, vol. 15, no. 1, pp. 72–101, 1904.

[64] L. Corneo et al., "Surrounded by the clouds: A comprehensive cloud reachability study," in *Proc. WWW*, 2021, pp. 295–304.


**Mengying Zhou** (Graduate Student Member, IEEE) received the B.Sc. degree in information security from Lanzhou University in 2019. She is currently pursuing the Ph.D. degree with Fudan University. Since 2018, she has been a Research Assistant with the Big Data and Networking (DataNET) Group, Fudan University. Her main research interests include network measurements, next-generation network architectures and protocols, and communication energy optimization.

**Tiancheng Guo** received the B.Sc. and M.Sc. degrees in computer science from Fudan University, China. His research interests include social computing, network routing, and urban mobility.

**Yang Chen** (Senior Member, IEEE) received the B.Sc. and Ph.D. degrees from the Department of Electronic Engineering, Tsinghua University, in 2004 and 2009, respectively. He is currently a Professor with the School of Computer Science, Fudan University, China, and the Vice Director of Shanghai Key Laboratory of Intelligent Information Processing. He has been leading the Big Data and Networking (DataNET) Group, since 2014. Before joining Fudan University, he was a Post-Doctoral Associate with the Department of Computer Science, Duke University, Durham, NC, USA. His research interests include social computing, Internet architecture, and mobile computing. He is serving as an editorial board member for *Transactions on Emerging Telecommunications Technologies* and an Associate Editor for *Computer Communications*. He served as an OC/TPC Member for many international conferences, including SOSP, SIGCOMM, WWW, IJCAI, AAAI, IWQoS, ICCCN, GLOBECOM, and ICC.



**Yupeng Li** (Member, IEEE) received the Ph.D. degree in computer science from The University of Hong Kong. He was a Post-Doctoral Research Fellow with the University of Toronto. He is currently an Assistant Professor with Hong Kong Baptist University. His research interests include network science and, in particular, algorithmic decision-making and machine learning problems, which arise in networked systems, such as information networks and ride-sharing platforms. He is also excited about interdisciplinary research that applies robust algorithmic techniques to edging problems. He has been awarded the Rising Star in Social Computing Award by CAAI and the Distinction of Distinguished Member of the IEEE INFOCOM Technical Program Committee. He serves on the technical committees of some top conferences in computer science. He has published articles in prestigious venues, such as IEEE INFOCOM, ACM MobiHoc, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, and IEEE/ACM TRANSACTIONS ON NETWORKING. He is a member of ACM.

**Meng Niu** (Member, IEEE) received the B.Sc. and Ph.D. degrees from Beijing University of Posts and Telecommunications in 2015 and 2021, respectively. He is currently with Huawei Technologies Company. His current research interests include future Internet architectures and routing protocols.

**Xin Wang** (Member, IEEE) received the B.Sc. degree in information theory and the M.Sc. degree in communication and electronic systems from Xidian University, China, in 1994 and 1997, respectively, and the Ph.D. degree in computer science from Shizuoka University, Japan, in 2002. He is currently a Professor with Fudan University, Shanghai, China. His main research interests include quality of network service, next-generation network architectures, mobile Internet, and network coding.

**Pan Hui** (Fellow, IEEE) received the bachelor's and M.Phil. degrees from The University of Hong Kong and the Ph.D. degree from the Computer Laboratory, University of Cambridge. He is currently a Professor of computational media and arts and the Director of the Center for Metaverse and Computational Creativity, The Hong Kong University of Science and Technology (HKUST). He is also the Nokia Chair in Data Science with the University of Helsinki. He taught with the Department of Computer Science and Engineering, HKUST, from 2013 to 2021, before moving to the Computational Media and Arts Thrust as a Founding Member. From 2012 to 2016, he was an Adjunct Professor of social computing and networking at Aalto University. He is an International Fellow of the Royal Academy of Engineering. He has founded and chaired several IEEE/ACM conferences/workshops. He has served as the track chair, a senior program committee member, an organizing committee member, and a program committee member for numerous top conferences, including ACM WWW, ACM SIGCOMM, ACM Mobisys, ACM MobiCom, ACM CoNext, IEEE Infocom, IEEE PerCom, IEEE ICNP, IEEE ICDCS, IJCAI, AAAI, UAI, and ICWSM. He served as an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING from 2014 to 2019 and IEEE TRANSACTIONS ON CLOUD COMPUTING from 2014 to 2018, and a Guest Editor for various journals, including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC), IEEE TRANSACTIONS ON SECURE AND DEPENDABLE COMPUTING, *IEEE Communications Magazine*, and *ACM Transactions on Multimedia Computing, Communications, and Applications*. He is an ACM Distinguished Scientist and a member of Academia Europaea (Academy of Europe).