Li, Xuebing; Chen, Yang; Zhou, Mengying; Guo, Tiancheng; Wang, Chenhao; Xiao, Yu; Wan, Junjie; Wang, Xin

Artemis

Published in:
IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS

DOI:
10.1109/TPDS.2022.3207189

Published: 01/12/2022

Document Version
Peer reviewed version

Please cite the original version:
Li, X., Chen, Y., Zhou, M., Guo, T., Wang, C., Xiao, Y., Wan, J., & Wang, X. (2022). Artemis: A Latency-Oriented Naming and Routing System. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, *33*(12), 4874-4890. https://doi.org/10.1109/TPDS.2022.3207189

Artemis: A Latency-Oriented Naming and Routing System

Xuebing Li, Yang Chen, *Senior Member, IEEE,* Mengying Zhou, Tiancheng Guo, Chenhao Wang, Yu Xiao, *Member, IEEE,* Junjie Wan, and Xin Wang, *Member, IEEE,*

Abstract—Today, Internet service deployment is typically implemented with server replication at multiple locations. Domain name system (DNS), which translates human-readable domain names into network-routable IP addresses, is typically used for distributing users to different server replicas. However, DNS relies on several network-based queries and the queries delay the connection setup process between the client and the server replica. In this paper, we propose Artemis, a practical low-latency naming and routing system that supports optimal server (replica) selection based on user-defined policies and provides lower query latencies than DNS. Artemis uses a DNS-like domain name-IP mapping for replica selection and achieves low query latency by combining the name resolution process with the transport-layer handshake process. In Artemis, all of the server replicas at different locations share the same anycast IP address, called Service Address. Clients use the Service Address to establish a transport-layer connection with the server. The client's initial handshake packet is routed over an overlay network to reach the optimal server. Then the server migrates the transport layer connection to its original unicast IP address after finishing the handshake process. After that, service discovery is completed, and the client communicates with the server directly via IP addresses. To validate the effectiveness of Artemis, we evaluate its performance via both real trace-driven simulation and real-world deployment. The result shows that Artemis can handle a large number of connections and reduce the connection setup latency compared with state-of-the-art solutions. More specifically, our deployment across 11 Google data centers shows that Artemis reduces the connection setup latency by 39.4% compared with DNS.

Index Terms—service discovery, name resolution, overlay routing, anycast

1 Introduction

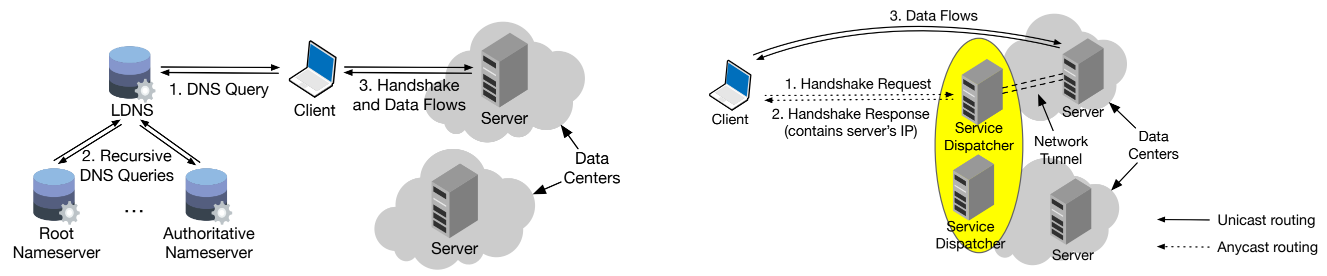
Low latency is a critical requirement for today's Internet services. Amazon found that a reduction of 100ms in page load time (PLT) contributes to an increment of 1% in revenue [1]. Similarly, Google reported that a 2s delay might cause a 4.3% loss in revenue per visit [2]. Servers are expected to be deployed close to the clients to reduce network latency. One of the solutions is to place replica servers worldwide so that the clients can always connect to their nearby servers. In the scenario where a set of replica servers sharing the same token (e.g., domain name and IP address) are available at multiple locations, it is essential to take into account the difference in the latency between the client and each replica, when deciding which replica to select for serving a request. Such a process of replica selection is called **service discovery**.

Today's service discovery mechanisms are either DNS-based or anycast-based. The DNS-based method redirects clients to replica servers by returning different IP addresses associated with the same domain name. This method provides flexibility for domain names. However, it causes significant latencies because the name lookup process requires recursive network-based queries from the client to a group of name servers, as shown in Fig. 1a. A previous measurement study showed that the DNS query latency typically ranges from 1ms to 5s [3], which consumes up to 13% of the PLT when browsing the Internet [4]. The anycast-based method selects replica servers based on minimal routing hops. The major drawback is the lack of application-level controls, e.g., load balancing [5]. Although DNS is not being used for replica selection in anycast routing, DNS is still generally required for name lookup [6], meaning that the latency caused by DNS name lookup cannot be ignored. To solve these challenges, we aim at developing a service discovery mechanism that would utilize the advantages of both anycast and DNS to shorten the overall latency.

In this paper, we propose ***Artemis***, a novel latency-oriented naming and routing system for service discovery. Artemis supports 1) DNS-like name resolution, where all replica servers are exposed to the clients as a single human-readable name, 2) optimal replica selection, where the discovered server is optimal for a client based on a configurable selection criterion, and 3) low service discovery latency, where name resolution does not cost additional query delay. Since Artemis is a service discovery solution rather than

- *Xuebing Li is with the School of Computer Science, Fudan University, China, the Shanghai Key Lab of Intelligent Information Processing, Fudan University, China, and the Department of Communications and Networking, Aalto University, Finland.*
  E-mail: xbli16@fudan.edu.cn
- *Yang Chen, Mengying Zhou, Tiancheng Guo, Chenhao Wang, and Xin Wang are with the School of Computer Science, Fudan University, China, and the Shanghai Key Lab of Intelligent Information Processing, Fudan University, China.*
  E-mail: {chenyang, myzhou19, tcguo20, xinw}@fudan.edu.cn, chenhaowang21@m.fudan.edu.cn
- *Yu Xiao is with the Department of Communications and Networking, Aalto University, Finland.*
  E-mail: yu.xiao@aalto.fi
- *Junjie Wan is with the Huawei Technologies Co. Ltd., China.*
  E-mail: wanjunjie2@huawei.com

- *Manuscript received April 4, 2021; revised January 4, 2022 and June 15, 2022; accepted August 30, 2022. (Corresponding author: Yang Chen)*



(a) DNS (b) Artemis

Fig. 1: Workflow of DNS and Artemis

a global unique naming system, e.g., DNS, we define the name space as any lexical phase to avoid the conflict with the widely-used DNS name space. In the following of this paper, we use "DNS name" and "Artemis name" to represent the name used in DNS and Artemis, separately. Because of the isolation with the DNS name space, Artemis is targeting application scenarios supporting dual name spaces. For example, in mobile apps, developers can hard-code the logic of switching between DNS and Artemis in the client code when accessing the app servers deployed in Artemis. For demonstration, we by default choose the optimal replica based on two commonly used performance metrics, i.e., end-to-end latency and server load. Our evaluation verifies the commonly used assumption that the delays occurred outside data centers (i.e., between clients and data centers) dominate the end-to-end delay [7], [8], [9]. Therefore, this work focuses on shortening the above-mentioned routing-induced latencies.

The architecture of Artemis is illustrated in Fig. 1b. We introduce an overlay network composed of a set of Service Dispatchers in different data centers. A Service Dispatcher works as a proxy for all the replicas located within the same data center. A Service Dispatcher is identified with Service Address, an anycast IP address composed of a predefined anycast prefix and a network suffix calculated from the Artemis name. It accepts the clients' packets and forwards each request to the optimal replica through overlay routing, which is determined by customizable selection criteria, e.g., minimal end-to-end latency and low server load. Instead of forwarding the packets directly to an optimal replica as done in [10], we implement redundancy routing into the overlay network of service dispatchers to allow the same handshake request to be sent to multiple replicas simultaneously. The replica processes the handshake request and generates a response to finish the connection setup. The server embeds its unicast IP address in the handshake response so that the subsequent packets are transmitted directly between the client and the server, reducing the load of the overlay network. This process is called late binding.

Artemis is compatible with any transport layer protocol that supports late binding, e.g., QUIC [11], M-TCP [12], etc. In this paper, we build a prototype of Artemis based on the state-of-the-art transport protocol QUIC and deploy it on Google Cloud. We choose QUIC because it allows changing the server's IP address without interrupting a connection, making it easier to implement late binding.

The key contributions of this paper are summarized below:

- We propose Artemis, a novel latency-oriented naming and routing system. According to our evaluation of a real-world deployment, Artemis shortens the connection setup latency (i.e., 39.4% reduction in the handshake latency) compared with DNS-based solutions since it removes the need for name resolution through dedicated query packets. Compared with anycast-based solutions, Artemis reduces the transmission latency by 25.2% by supporting customized routing policies at the stage of name resolution.
- We introduce Service Dispatchers, which bind a client with its best suitable replica server based on custom end-user-mapping policies. We present the late binding mechanism, which binds a client with a server with an anycast address, overcoming the uncertainty caused by anycast routing. The late binding supports the client-side cache of the client-server binding and supports invalidation of the cache from the server-side.

A preliminary version of this paper has been published in [10]. The new contributions mainly come from the implementation and feasibility analysis of the real-world deployment of Artemis. As a representative application scenario of Artemis, Cloud services will benefit from deploying Artemis by 1) an easy way to manage IP addresses, 2) a fast name lookup service, 3) and native support in server duplication for scalability. The real-world evaluation in Section 5.3 demonstrates the feasibility of deploying Artemis in commercial clouds, e.g., Google Cloud, and its effectiveness in providing low-latency naming and routing for replica servers serving global clients.

The following of this paper is organized as below. Section 2 gives an overview of different replica selection strategies. Section 3 and Section 4 describe the system design and implementation, respectively. The system evaluation is presented in Section 5. A brief discussion is given in Section 6 before concluding our work in Section 7.

2 Related Work

In this section, we review the previous works related to three categories of server selection strategies. A comparison between Artemis and state-of-the-art solutions is summarized in Table 1.

Approach	Methodology	Server Load Awareness	Connection Setup Latency	Server Selection Result
FastRoute [8]	Anycast & DNS	Yes	High, with DNS latency	Suboptimal
Google load balancing [13]	Anycast	No	Low	Suboptimal
Akamai [14]	DNS	Yes	High, with DNS latency	Optimal
Artemis (this paper)	Anycast	Yes	Low	Optimal

TABLE 1: Comparison of related works on server selection.

2.1 DNS-Based Solutions

DNS is an Internet service that maps human-readable names into machine-readable IP addresses [15], [16]. In DNS-based solutions, authoritative nameservers monitor the condition of application servers, e.g., server load, and keep track of the routing information from the servers to the rest of the Internet, e.g., end-to-end latency. According to the collected information and the service discovery policy, the authoritative nameservers determine the best replica server for each client. Akamai [14] adopts a DNS-based solution, with the support of EDNS Client Subnet (ECS) [17], in its content delivery network (CDN) for load balancing. Google employs similar technology in its fronted serving architecture [18]. The advantage is that the nameserver has an overview of the whole system. It can select any replica for the client and take application-level information into account. However, the disadvantage is that the DNS query brings additional latency to the connection setup process.

Artemis combines anycast's fast connection setup with DNS's controllability. As a result, Artemis supports low-latency connection setup while performing optimal server selection based on customizable policies.

2.2 Anycast-Based Solutions

Anycast addressing is a process of one-to-many association where packets are routed to the end-host within a group identified with the same IP address. It utilizes the Border Gateway Protocol (BGP [19]), the de-facto standard inter-domain routing protocol on the Internet, to select the shortest routes to reach a destination. In most cases, the selected end-host is close to the client, whereas in some cases, the destinations are located far away, e.g., thousands of kilometers [20], from the client.

Anycast-based server selection has been used by DNS servers [21], [22] and CDNs including Microsoft Azure [23], Edgecast [24], CloudFlare [6], and Google Cloud [13], [25]. Replicas are assigned with the same anycast address, meaning that the replica selection is up to the anycast routing policy in use. Cloudflare, for example, implements its replica selection purely on anycast, where a client connects to a replica server directly with an anycast address. Such server selection method is straightforward but does not support customized selection criteria, e.g., server load. Moreover, in rare cases [26], a connection may be interrupted because anycast does not guarantee the server affinity, i.e., the client's packet may arrive at a different replica server that it communicated with previously.

FastRoute [8] is proposed as a load-aware anycast routing platform. It deploys DNS servers co-located with replica servers for name lookup and traffic redirection. The servers are deployed following a tree structure consisting of several layers. When the nodes at the lower layers (starting from leaf nodes) become heavily loaded, they redirect new incoming connections to the nodes at the higher layers. In this way, FastRoute is load-aware and guarantees server affinity. But it delays the connection setup process because of the use of DNS.

Artemis achieves fast service discovery via anycast routing. Compared with legacy anycast routing, Artemis supports customized replica selection criteria via overlay routing and solves the problem of server affinity via late binding. Because both overlay routing and late binding are integrated into the connection setup process, Artemis does not introduce additional latency compared with legacy anycast routing.

2.3 Application Layer Solutions

Application layer solutions usually reselect servers after the client has connected to one of them [27], [28], [29]. For example, HTTP status code 3xx for redirection is commonly used in web services [27], [28]. Some others use custom protocols for redirection in the application layer [29]. The application layer solutions take into account the status of the clients, e.g., locations. However, due to the additional latency caused by redirection, it is not necessarily ideal for latency-sensitive Internet applications.

2.4 Overlay Networks

Overlay is a classic and powerful technique of forwarding packets with custom policies and is commonly used for optimizing packet routing. For example, BDS [30] utilizes an inter-datacenter overlay to accelerate the speed of data duplication. In video transmission, overlay is used to reduce the transmission delay by optimizing the routing path [31], [32]. Although these applications have shown overlay's efficiency in optimizing the quality of service (QoS) in various scenarios, its overhead is not negligible. Firstly, the overlay routing is usually implemented in the software. The packet transmission inside an overlay node involves several traverses through the network stack, which imposes overhead in terms of both throughput and latency [33]. Secondly, the packet loss ratio in overlay networks is larger than in the physical networks because of the unreliability in the overlay nodes, which challenges the QoS of the applications running on top of it [34].

In Artemis, we utilize overlay for supporting customizable routing policies. Meanwhile, we try to avoid the overlay's side effects. To minimize the additional delay caused by the overlay, we reduce the number of packets going through it, i.e., only the first packet in a connection is routed through the overlay. To prevent packet loss, we introduce redundancy routing that duplicates packets to achieve higher reliability. Moreover, as discussed in Section 6.2, the use of overlay in Artemis successfully addresses many challenges in anycast routing.

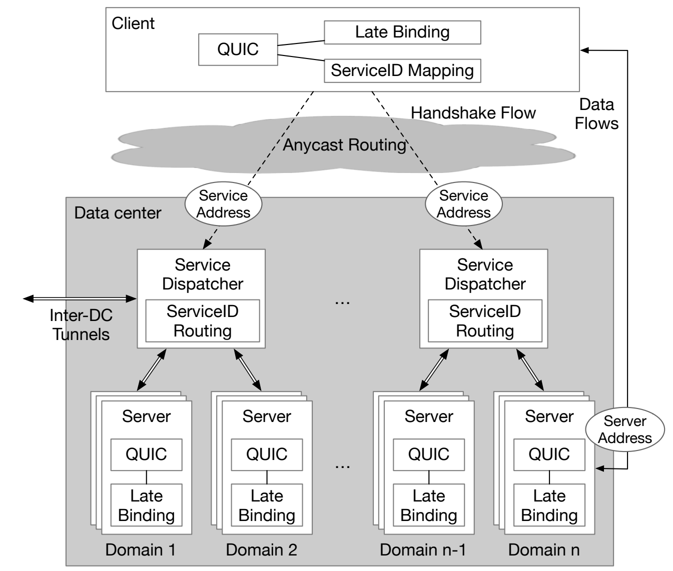


Fig. 2: Artemis architecture within a data center. A client connects to a Service Dispatcher through the Service Address. The Service Dispatcher forwards the handshake request to the optimal replica server. The replica server migrates its IP address from the Service Address to its unicast IP address (the server address in the diagram) for connection.

3 Design

In this section, we give an overview of Artemis in Section 3.1 and describe each module in detail from Section 3.2 to 3.5.

3.1 Overview

The architecture of Artemis is shown in Fig. 2. We introduce an overlay network between the clients and the servers for redirecting clients' handshake requests. The overlay network is composed of several software routers, called Service Dispatchers. They are deployed at different data centers and are interconnected via inter-datacenter tunnels for packet forwarding. All of the Service Dispatchers advocate the same anycast network prefix via BGP announcements so that a client's packet may reach any data center via anycast routing. In each data center, the replica servers are assigned IP addresses within the IP subnet advocated by the Service Dispatcher. We use a hash-based method to ensure that different replicas for the same Artemis name have the same anycast IP address, which is called Service Address.

The connection setup process in Artemis is composed of three steps. 1) ServiceID mapping. The client firstly calculates the Service Address according to the Artemis name and sends a handshake request to the calculated address. 2) End-user mapping and ServiceID routing. Upon the request's arrival via anycast routing, the Service Dispatcher in question selects one data center based on the customizable criteria and forwards the request to the Service Dispatcher in that data center. Ideally, the request should be forwarded to the nearest data center with service replicas deployed. Then, the Service Dispatcher forwards the request to the replica server inside its data center. 3) Late binding. The replica server processes the request and sends back a handshake response with its unicast IP address embedded. After receiving the handshake response, the client migrates the connection with Service Address to that with the replica server's unicast IP address.

After the connection setup, the server's unicast IP address is cached at the client to accelerate future connections. In case the server is not suitable for the client anymore, the server can invalidate the client-side cache by refusing the client's new connection.

3.2 ServiceID Mapping

The ServiceID mapping module resides in the clients and the Address Managers. The Address Managers are deployed inside every data center and are responsible for allocating Service Addresses to the replica servers. The ServiceID mapping module takes Artemis name as input and generates Service Addresses as output. A generated Service Address is composed of two parts: a service prefix and a ServiceID.

The service prefix is the same as subnet ID in IP address space and is advocated by BGP routers within all of the data centers, making it an anycast subnet. If a client wants to use Artemis, it needs to acquire the network prefix from the service provider beforehand. The network prefix will be used as a unique identity of an instance of Artemis. Anycast routing provides a simple but efficient solution to direct a client to its nearest data center. Artemis takes advantage of this feature as previous works described in Section 2.2.

ServiceID is designed for load balancing via static hashing. It is generated by the Artemis name according to a predefined hash function. Given an Artemis name, we generate a SHA512 [35] hash of it and choose the first N_{sid} bits as the ServiceID, where N_{sid} is the length of the ServiceID in bits. The value of N_{sid} is flexible and can be determined by the Artemis providers according to their requirements. Generally, the longer the ServiceID is, the more allocable addresses there are. However, a larger address space also increases acquisition and maintenance costs. A particular case is setting N_{sid} to 0. The service address will be a fixed IP address, assigned to the only Service Dispatcher deployed at each data center.

With the design of Service Address, we manage to achieve three goals. Firstly, the destination IP address is derived from the Artemis name without any network-based queries (e.g., DNS query), which is the primary reason why Artemis achieves lower name resolution latency than DNS. Secondly, the Service Dispatcher with the calculated service address is highly likely to be located in a nearby data center, following the principle of geolocation-based anycast routing. Thirdly, the ingress traffic is distributed among all Service Addresses, meaning that we can increase N_{sid} to raise the number of Service Dispatchers and therefore reduce the load of each Service Dispatcher.

DNS name space. Although Artemis technically supports any lexical phrases on name resolution, it is challenging to keep the DNS name space because the latter strictly forces third-party services to synchronize their name records with the DNS root servers [36], [37]. The synchronization delays Artemis in name resolution and requires additional standardization work. So, we left the support of DNS name space in Artemis as future work and focused on a technical solution that translates lexical server names into IP addresses.

3.3 End-User Mapping

The end-user mapping module is a distributed algorithm running on each Service Dispatcher to implement the replica selection criteria. We aim to find a replica server with the lowest latency to the client in question since low latency is a common requirement of Internet services [14]. Note that Artemis can support other customized policies as well.

From the handshake request, we get the information on where the request comes from (the client's source address) and what service the client is requesting (the Artemis name)[1] .

We use a database to provide data storage of high scalability and reliability. The database contains two tables named *ServiceDeployment* and *LatencyMeasurement*. Suppose there exists at least one replica server in a data center. In that case, a record indicating the Artemis name associated with the replica server and the data center identity is added to the *ServiceDeployment* table in the format of (Artemis name, data center). The *LatencyMeasurement* table records the RTT between every pair of data center and client subnet, which is measured by active probing periodically. A record (subnet, data center, RTT) is updated in the *LatencyMeasurement* table when one round of measurement completes[2]. The database is deployed in the primary-secondary mode to provide high accessibility. The primary node is statically configured in one of the Service Dispatchers, and the secondary node runs on all of the other Service Dispatchers. To ensure the efficiency of data synchronization, we allow the write operation on the primary node only.

The replica server selection procedure is as follows. Firstly, given a client subnet and a target Artemis name, the Service Dispatcher queries the *ServiceDeployment* table to find out the data centers where replica servers with the given Artemis name exist. Then, based on the previous result set, it queries the *LatencyMeasurement* table to find out the nearest data center to the client. Finally, different types of packet routing are applied based on the query results.

3.4 ServiceID Routing

The ServiceID routing module is designed to forward the packets to the optimal replica server for each client. It comprises two parts, i.e., the inter-DC routing (Section 3.4.2), which directs packets to the optimal data center, and the intra-DC routing (Section 3.4.1), which directs packets to a proper server within the selected data center. Furthermore, we use redundancy routing (Section 3.4.3) to reduce the tail latency in Artemis, caused by the dynamics as well as the unreliability of the Internet routing. The packet path is shown in Fig. 3.

1. QUIC enforces the handshake request to contain the server's name, as the value of an extension in TLS 1.3 called server name indication (SNI) [38]

2. The active measurement method is designed for IPv4 and does not work for IPv6 because of the latter's large address space. Considering that IPv4 is still dominating the Internet traffic [39], we are targeting IPv4, and the support of IPv6 is left for future work. To support IPv6, we may use passive measurement to limit the measurement scope to active clients instead of the whole Internet [40].

Condition		Action
in_port	optimal_check	output
N_{bu}	true	TB_{si}
N_{bu}	false	TB_{bi}
TB_{si}	/	N_{bu}
TB_{bi}	/	TB_{si}

TABLE 2: The routing table of a Service Dispatcher.

3.4.1 Inter-DC Routing

At first, the client sends a handshake request to the Service Address (step 1 in Fig. 3). Every data center has its own Service Dispatchers to handle the Service Addresses and the request may reach a Service Dispatcher inside any data center. The inter-DC routing infrastructure is designed to route the handshake request from where it is received, $DC_{anycast}$, to the optimal data center, $DC_{optimal}$, which is determined by the end-user mapping module.

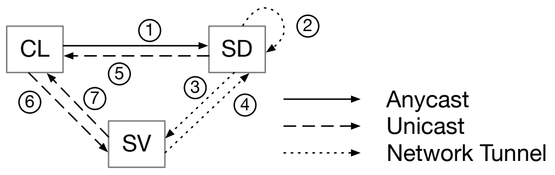
On receiving the handshake request, the Service Dispatcher extracts the Artemis name from the request and uses the end-user mapping module to find $DC_{optimal}$. If $DC_{optimal}$ refers to a different data center, the handshake request is forwarded through the network tunnels to $DC_{optimal}$ (step 2 in Fig. 3). After reaching $DC_{optimal}$, intra-DC routing is conducted to forward the request further to a replica server.

3.4.2 Intra-DC Routing

The next step is to dispatch the handshake request from the Service Dispatcher to the corresponding replica server within the data center. A database table called *ReplicaDeployment* is created to record the information of all replica servers. Unlike the tables in Section 3.3, *ReplicaDeployment* is stored locally without synchronization in the Service Dispatchers because the data is used only inside the data center. Whenever a new server is registered to a domain, a new entry (domain, tunnel name) will be inserted into the table. The entry will be deleted when the server stops serving the domain.

Upon receiving the handshake request, the Service Dispatcher queries the *ReplicaDeployment* table and selects a replica server for the domain. We perform random selection among server replications to evenly distribute the requests among them. Then, the Service Dispatcher forwards the handshake request to the selected replica server through a network tunnel (step 3 in Fig. 3). Afterward, the replica server receives the original handshake request from the client. After processing, the replica server generates a handshake response and sends it back to the client. To avoid being blocked by the client's firewalls, the response packet is delivered back to the client through the original route (steps 4 and 5 in Fig. 3). From the client's view, it has established a connection with the Service Dispatcher, which is a proxy for the replica server. Then, the connection is migrated to the server's unicast IP address with the help of the late binding module. Finally, the client and the server can communicate directly with unicast IP addresses (steps 6 and 7 in Fig. 3).

To put it all together, the routing table of the Service Dispatcher is shown in Table 2. TB_{bi} means the network tunnels towards the other Service Dispatchers, and TB_{si} means network tunnels towards the replica servers. N_{bu} refers to



No.	Src	Dst	No.	Src	Dst
1	CL.IP	SA	5	SA	CL.IP
2	CL.IP	SA	6	CL.IP	SV.IP
3	CL.IP	SA	7	SV.IP	CL.IP
4	SA	CL.IP			

Fig. 3: Packet paths of handshake flow and data flow in Artemis. The table on the right shows various packet headers. Symbols in block letters represent different roles in Artemis. CL is the client, SD is the Service Dispatcher, and SV is the server. SA stands for Service Address and is the IP address of the Service Dispatcher. CL.IP is the IP address of the client. It may be the router's IP address if the client is behind a NAT. SV.IP is the unicast IP address of the server. The network tunnels transfer packets between two hosts bidirectionally.

the port assigned with the Service Address in a Service Dispatcher. $optimal\_check$ indicates whether $DC_{anycast}$ equals to $DC_{optimal}$.

Artemis gains three advantages by the design of ServiceID routing. Firstly, the routing policy knows rich elements, including the system state, the network performance, and the application-level information. Secondly, all of the data mentioned above are persisted in the database. The Service Dispatchers are stateless and can recover from failures quickly. Thirdly, network tunnels enable us to complete the routing by only modifying a small part of the replica servers' network stacks. More details about the implementation are given in Section 4.1.

3.4.3 Redundancy Routing

With the design of inter-DC routing and intra-DC routing, Artemis can route a client's packet to its optimal replica server. In this paper, the optimal rule is defined as choosing the one which would not be overloaded and would provide the lowest end-to-end latency. However, there are cases where inter-DC routing fails to choose the replica server of the lowest end-to-end latency. Firstly, the result of end-user mapping may be outdated. Although the state-of-the-art network measurement tools can scan the Internet within 5 minutes [41], it is not practical to update the Inter-DC routing tables at such high frequency due to the overhead of synchronizing a massive amount of entries within a globally deployed cluster. We set the update frequency as once per day in Artemis. Secondly, packet loss is unavoidable on the Internet and has been found as a significant cause of tail latency in DNS [42]. In Artemis, a lost handshake packet would significantly delay the handshake process due to the time spent on waiting for retransmission. With the help of redundancy routing, the lost packet affects only one replica server, and nearby servers receiving the request can still establish a connection with the client without waiting for retransmission. Thirdly, a malfunctioning, e.g., overloaded, replica server may fail to respond to a client's handshake request. Nearby servers can automatically replace the malfunctioning one by actively responding to the client's handshake request.

At the stage of inter-DC routing, instead of only forwarding the client's handshake request to the optimal data center, we duplicate the handshake request and forward them to the top X data centers, where X is a configurable variable starting from 1. With redundancy routing, X replica servers are receiving and responding to the client's handshake request. The client accepts the first received handshake response and discards the handshake response from the other replica servers.

Overhead of the redundant packets. The most significant side effect of duplicating packets on the Internet is the increase in data traffic, thus causing additional workload to all participants of the packet transmission system. We carefully design the redundancy routing to mitigate the side effect of duplicating packets. Firstly, we only duplicate handshake packets, which make up a minority of the Internet packets. So, the newly introduced traffic is much smaller than the existing traffic handled by the Internet infrastructure. Secondly, the duplicated packets cause more traffic to the intra-DC routing than to the inter-DC routing. It is comparatively easy to be handled by the Artemis infrastructure because the intra-DC routing only deals with regional packets. Thirdly, owing to QUIC's security design, the replica servers can avoid wasting resources in handling the half-open connections caused by the duplicated handshake packets, which is an important security feature of handling dynamic denial of service (DDoS) attacks [43]. Finally, Artemis is flexible at dealing with the side effect of the redundant packets by setting a proper value of X.

3.5 Late Binding

The late binding module is to bind a client with the selected replica server. When initiating a connection, the client sends a handshake request to the Service Address, and the handshake request reaches a replica server through the ServiceID routing. After processing the handshake request, the server generates a response and sends it back to the client. The one round trip of handshake packets finishes the handshake process, along with the service discovery process. To reduce the Service Dispatcher's load, all of the following packets are delivered directly between the client and server without going through the overlay network. So, we design the late binding module to enable a server to migrate the connection from Service Address to its unicast IP address.

QUIC already supports the connection migration on completion of the handshake, called the server's preferred address [11]. The server can accept connections on one IP address and transmit data from a more preferred IP address shortly after the handshake. The mechanism is described below.

Both the handshake request and the handshake response can carry transport parameters. The transport parameters are defined as key-value pairs. The key is of integer format and the value is of binary format with flexible length.

The server embeds its preferred address inside the handshake response with the key of *preferred_address* (0x000d). The client extracts the preferred address and sends further packets to the server's new address immediately. The late binding is fast because it is integrated into the handshake process and does not cause extra latency. Since QUIC has native support in mobility, changing the server's IP address does not complicate either the server's or the client's state machine.

Besides, we use a cache mechanism of the server's preferred address on the client side. A key named service_ttl (0xff00) is registered as a new transport parameter. The value is in the integer format, representing how many seconds the server's preferred address should be cached on the client side. As long as the time to live (TTL) is not expired, the client is allowed to make new connections to the service replica via the replica's unicast IP address directly. Otherwise, the client must query for the optimal server again by using the Service Address.

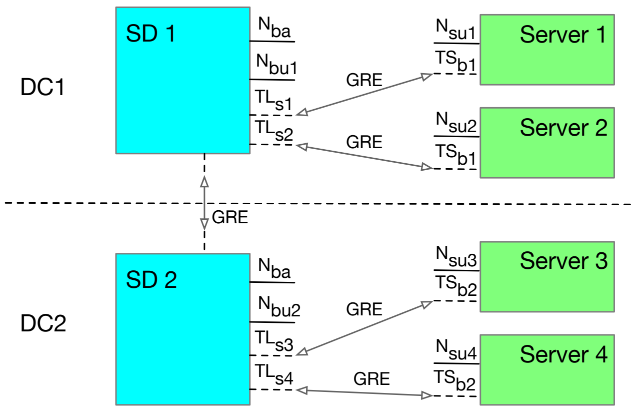


Fig. 4: An example scenario of Artemis deployment across two data centers. All of the servers serve the same domain. DC stands for data center, and SD stands for Service Dispatcher. Solid lines represent physical NICs, dashed lines represent virtual interfaces, and lines with arrays represent GRE tunnels.

4 Implementation

Our implementation is based on the cloud infrastructure where replica servers are deployed at several geo-distributed data centers to serve global clients. At each data center, there are one or more Service Dispatchers, which are software routers with support of overlay routing, and a group of servers that work for their dedicated names. The following of this section is organized as below. Section 4.1 presents the implementation of overlay routing, and Section 4.2 presents the modification of QUIC, including both server-side and client-side changes. In Section 4.3, we demonstrate the deployment of Artemis on Google Cloud as a cloud-based evaluation platform.

4.1 Tunnels and Packet Forwarding

Generic Routing Encapsulation (GRE) is a protocol for encapsulating data packets of one routing protocol inside packets of another protocol. Artemis utilizes Layer-3 GRE tunnels to transfer IP packets between service dispatcher and replica server or between service dispatchers. In practice, Open vSwitch (OVS) [44] is used for tunnel establishment and packet forwarding.

Fig. 4 illustrates an example scenario of Artemis deployment across two data centers. A Service Dispatcher is equipped with two physical NICs, namely N_{ba} and N_{bu}. N_{ba} is assigned with one or more anycast IP addresses (ServiceIDs) for the anycast routing. N_{bu} is assigned with a unicast IP address for establishing GRE tunnels. The server is equipped with only one physical NIC, namely N_{su}. It is assigned with a unicast IP address to establish GRE tunnels and perform data transmissions. Note that the Service Dispatcher does not need multiple NICs because multi-homing, where multiple IP addresses are assigned to the same NIC, can provide the same functionality. We choose multiple NICs in implementation because our evaluation platform, i.e., Google Cloud, does not support multi-homing.

GRE tunnels and virtual interfaces are created on the Service Dispatchers and the replica servers. A Service Dispatcher establishes GRE tunnels to the Service Dispatchers in other data centers, namely TL_{bi} for the ith data center, and the replica servers in the same data center, namely TL_{si} for the ith replica server. On each service dispatcher and replica server, a virtual interface is created for each GRE tunnel to provide the reading and writing access to the applications (i.e., QUIC servers). The OVS is configured to forward all packets from virtual interfaces to the paired endpoint through GRE tunnels. On the servers, TS_{bi} is the corresponding endpoint of TL_{si}. It connects the Service Dispatcher (denoted as bi) and is configured with the same IP address as ServiceID, which is also the IP address of N_{ba}.

The Service Dispatcher does not generate any extra packets. It simply routes received packets to their proper destinations. The routing policy is implemented in two parts. For the ingress packets arriving from N_{ba} and TB_{bi}, we implement a module to parse and route QUIC handshake packets as follows. Firstly, it creates a raw socket for each interface, including physical NICs and virtual interfaces. The sockets are bound to the interfaces via the socket option SO_BINDTODEVICE. Secondly, it uses the created sockets to receive IP packets from the NICs and parses the QUIC handshake packets. According to the routing policies described in Section 3.4, the packets are forwarded to specific destinations directly by writing to the corresponding socket. The socket option IP_HDRINCL is set to prevent the system from filling in the IP headers. For the ingress packets arriving from TB_{si}, we create data flows in OVS to send out all packets received from the servers (with in_port equal to TB_{si}) to the Internet via N_{ba}. Since TS_{bi} is already configured with the ServiceID, which is of the same address as N_{ba}, the Service Dispatcher does not need to change any fields of the IP packets. In this way, the packets manage to bypass the source address check.

4.2 Modifications on the Network Stack

As mentioned in Section 3.5, we implement the "preferred address" feature of QUIC and the ServiceID mapping module on both the client-side and the server-side.

On the client-side, we add a new system call named *getaddrinfosid*, which takes the Artemis name as input and outputs Service Address, as an alternative to the system call named *getaddrinfo*, which is for DNS query. To use Artemis, the client application needs to change the DNS-based application code from invoking *getaddrinfo* to invoking *getaddrinfosid*, which may require recompilation or patch on the existing application. The code change is limited to the function name as both system calls have the same input and output format. We do not simply replace the implementation of *getaddrinfo* because the other applications that do not support Artemis should be able to use DNS for name resolution.

On the server-side, we modify the QUIC server to make it work on multiple interfaces with finer-grained control over the routing. The legacy QUIC server creates only one socket and listens on the UDP port 443. Upon receiving a packet, it extracts the packet source address and sets it as the IP address of the remote peer. This mechanism works on multiple interfaces by leaving the choice of the outgoing interface to the system routing table. We create a socket for each interface, including the physical NIC (N_{su}) and the GRE tunnels (TS_{bi}). The socket and the interface are bound with the socket option SO_BINDTODEVICE. Upon receiving a packet, the QUIC server records which socket it reads from and performs write operations on the same socket. Therefore, the responses can follow the same path back to the client. The late binding module utilizes these features by changing the socket from TS_{bi} to N_{su} when the client changes the remote peer from the Service Address to the server's IP address.

We find that almost all IETF QUIC implementations have supported the transport parameter *preferred_address*. However, none of them have implemented the logic of migrating the connection[3]. To the best of our knowledge, we are the first to implement this functionality. Our modification to the QUIC client and the QUIC server is based on ngtcp2[4].

4.3 Deployment of the Emulation Platform

We deploy the emulation platform as shown in Fig. 5, which is composed of application clients running over 15 AWS zones and application servers, as well as the Artemis infrastructure, running over 11 Google Cloud zones. Each AWS zone hosts a client machine running Ubuntu 18.04 LTS over 1 vCPU and 1 GB memory. Inside each Google Cloud zone, we deploy two machines, *router* and *server*, separately. Both run Ubuntu 18.04 LTS over 1 vCPU and 3.75 GB memory. The *server* works as an application server, and the *router* works as a Service Dispatcher. Ideally, the *router* is an SDN router integrated with the cloud routing infrastructure. However, it is impossible to modify routers in commercial clouds. So, we set up VxLAN[5] tunnels between the *router* and the *server* as described in Section 4.1 to simulate the *router* as a gateway to the *server*.

3. We have checked ngtcp2, Minq, mozquiz, picoquic, and quicly.

4. https://github.com/ngtcp2/ngtcp2

5. Google Cloud does not allow GRE tunnels, so we use Virtual Extensible LAN (VxLAN) for tunneling instead. Because we do not use any layer 2 feature provided by VxLAN, GRE and VxLAN are interchangeable in our deployment.

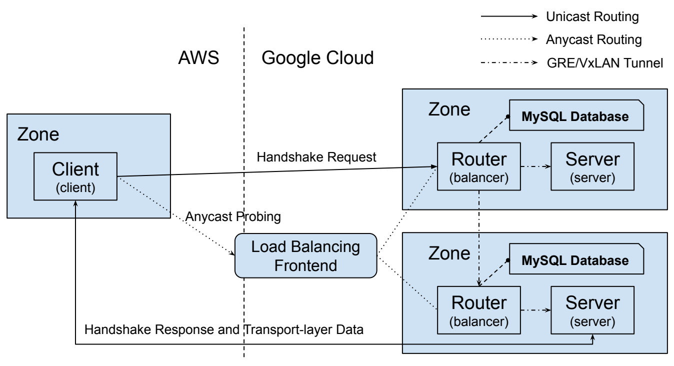


Fig. 5: Setup of the emulation platform. AWS is used to host the application clients. Google Cloud is used to host the Artemis infrastructure and the application servers. The load balancing frontend from Google Cloud is used to provide anycast IP addresses.

The other obstacle we face with Google Cloud is the lack of anycast address. Google Cloud can only assign unicast IP addresses to the VMs, whereas anycast addresses are required in the design of overlay routing. To solve this problem, we use Google's cloud load balancing[6] to simulate anycast routing. Google Cloud uses a single anycast IP to frontend all of the backend instances in the regions around the world. We assume that the affiliated frontend and backend are hosted at the same geolocation, meaning that the anycast IP address assigned to the frontend is logically equivalent to being associated with the backend server. We set up web servers in all regions as replicas and expose them with a single anycast IP address provided by the load balancing frontend. The web server is configured to return its region name on HTTP requests. In this way, the client can find out to which data center its request is forwarded via anycast routing. We call this procedure *anycast probing*.

With the implementation, the dataflow of a client-initiated connection is shown below.

1) A client uses *anycast probing* to find the data center routed by anycast addressing.
2) The client sends a handshake request to the router's IP address inside the found data center.
3) The router forwards the handshake request to a server, and the server completes the handshake process by returning a handshake response, inside which the server's IP address is embedded.
4) After the handshake procedure of Artemis, the client replaces the remote peer's IP address with the server's IP address.

5 Evaluation

We evaluate the performance of Artemis in three steps. Firstly, we conduct microbenchmarking to assess the scalability of Artemis. Secondly, we compare Artemis with the state-of-the-art via simulation, which is built on real-world measurements of DNS latency and anycast latency. Thirdly, we deploy Artemis on Google Cloud and analyze the performance impact of the geo-distribution of servers.

6. https://cloud.google.com/load-balancing

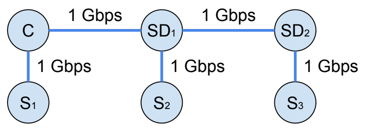


Fig. 6: Network topology and network parameters used for benchmarking: client (C), server (S), and Service Dispatcher (SD).

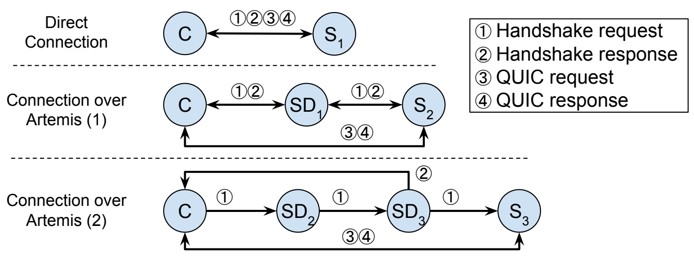


Fig. 7: Processes of connection establishment with and without Artemis. Each case contains a four-way communication.

5.1 Microbenchmarking

Experiment setup. We create an emulated network environment following the topology illustrated in Fig. 6. Each host, whether a client, a server, or a Service Dispatcher, is equipped with a 4-core 3.40 GHz Intel Xeon CPU and 16 GB memory, running Ubuntu 16.04 LTS (64-bit). All of the hosts are wired connected in a 1 Gbps local area network (LAN). The RTT between each pair of connected hosts is measured to be smaller than 1ms. Such a setup guarantees that the network would not become the bottleneck in the experiment.
Test cases and metrics. We design the following two sets of test cases to compare the performance with and without using Artemis and analyze the overhead of Artemis based on the comparisons.

- **Direct Connection:** The client establishes a QUIC connection with a server directly based on the server's IP address.
- **Connection over Artemis:** The client makes QUIC connections over Artemis.

We vary the number of concurrent connections and compare the latencies and hardware usages for both test cases. The measurement metrics are listed below.

- **Handshake latency**: the time spent between sending a handshake request and receiving a handshake response. It occurs once during the life cycle of a connection.
- **Connection setup latency**: handshake latency plus DNS query latency when DNS is in use. Otherwise, it is equal to handshake latency.
- **Transmission latency**: the RTT between a client and a server. It is the most important latency metric because it affects the transmission delay of all packets in a connection.
- **Query latency**: connection setup latency plus the time spent between sending a QUIC request and receiving a QUIC response, i.e., the overall latency of a short-lived connection. It is another important latency metric because connections with few packets were reported to make up about 48% of the network flows in a study on the campus network [45].
- **Hardware usages**: CPU usage and memory usage on each host.

There are two different scenarios in the case of Artemis, depending on whether the replica selected through anycast routing happens to be the optimal one according to the selection criteria. If it is the case, the client's handshake request only goes through a service dispatcher once. If not, the request would be redirected inside the overlay network, meaning that it must go through two service dispatchers. Based on the measurement result in Section 5.2, in 82.3% of the cases, the requests just need to go through one service dispatcher.
Experiment result As shown in Fig. 8, when the client connects directly with the server, the handshake latency, query latency, and memory usage increase linearly with the number of concurrent connections. The CPU usage reaches 30% when there are 1,000 concurrent connections.

For comparison, as shown in Fig. 9, the handshake latency and the query latency increase by 20.9ms and 27.3ms, respectively, when the packets need to go through service dispatchers. Our implementation is in user space. It receives packets from the kernel space, parses packets in the user space, and delivers packets back to the kernel space. This process can be accelerated if the program runs in kernel space, which is left for future work. Unlike the clients and the servers, the memory usage of the Service Dispatchers keeps steady when the number of concurrent connections increases. It is because Service Dispatchers are stateless. They do not need to maintain state machines for ongoing connections. The CPU usages of both clients and servers are almost the same in both cases. It validates that Artemis does not harm either the client's or the server's performance.
Scalability According to the microbenchmarking result, we conclude that Service Dispatcher will not be the bottleneck of the system because of the following three reasons:

- The stateless design of Service Dispatcher. Because of late binding, all of the packets after the handshake are transmitted between clients and servers directly. So, there is no need for Service Dispatcher to maintain connection states, making it easy to handle a large number of concurrent connections.
- The use of anycast. Artemis exposes Service Dispatcher via anycast IP addresses, which has been demonstrated as easy-to-use and efficient for load balancing, i.e., evenly distributing geo-distributed clients to nearby replica servers.
- Client-side cache. Like DNS, Artemis caches the name resolution result on the client to prevent repeated name lookup within a predefined TTL. It reduces the load of Service Dispatchers.

5.2 Simulation

In this subsection, we reuse the network topology shown in Section 5.1 but configure the RTTs among each host based on real-world measurement of end-to-end latencies from the

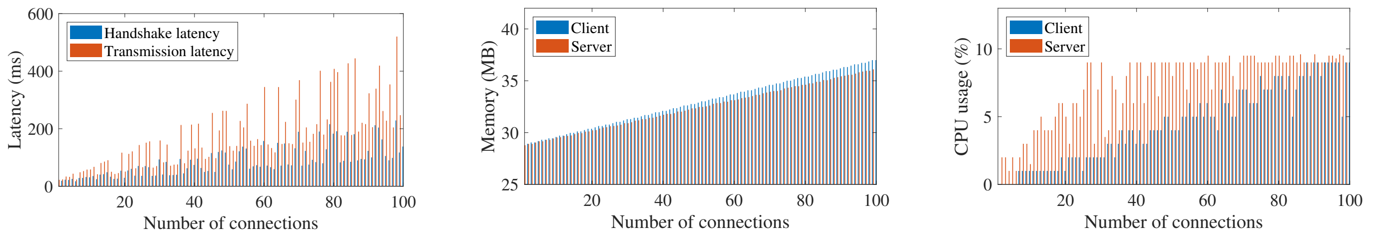


Fig. 8: Performance over different numbers of concurrent connections when the client connects to the server directly.

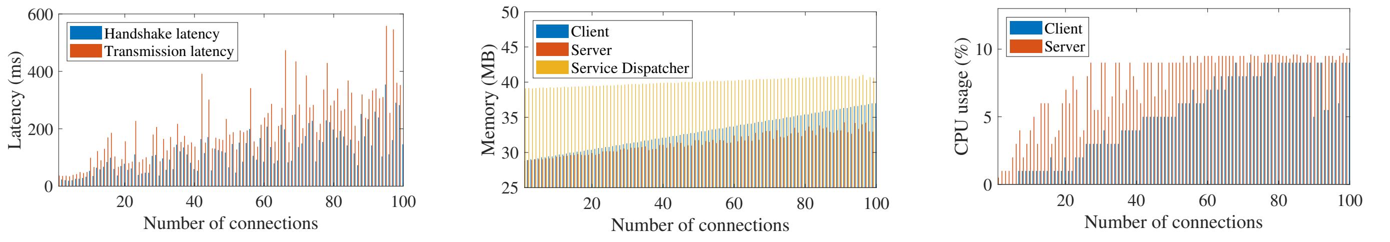


Fig. 9: Performance over different numbers of concurrent connections when the client connects to the server via Artemis.

Internet. Section 5.2.1 presents our method for measuring end-to-end latency on the Internet, and Section 5.2.2 shows our approach to simulating different test cases via the collected latency data and the evaluation results.

5.2.1 *Real-world latency measurement*

DNS latency. RIPE Atlas [46] is a global Internet measurement platform consisting of thousands of measurement devices. We use 500 RIPE Atlas probes as DNS clients to make DNS requests and measure the response delay. The authoritative name servers of the DNS names in query are hosted by Route53[7] as type A records, and the TTL of the records is set to 60 minutes. The DNS names queried from different DNS clients are within the same domain zone but have other subdomains, so that the DNS server can distinguish the clients from their queries. During the measurement, each client sends three consecutive DNS queries. The first query is for bootstrap and is of a different DNS name from the later two, which are of the same DNS name. The second query will encounter a cache miss because the DNS name has never been queried before. The third query will encounter a cache hit if and only if there is only one DNS query record from a client on the authoritative name server. In this way, we measure $latency_{hit}$ and $latency_{miss}$ as the DNS query latency on cache hit and cache miss, respectively. Notice that DNS cache miss may occur at different levels in the DNS hierarchy. For example, a miss in the name server (NS) record results in a recursive query to the top level domain (TLD) name server and to the authoritative name server. On the contrary, a miss in the A record requires only one query to the authoritative name server. The cache miss encountered in the measurement is the later case. Therefore, $latency_{miss}$ is the lower bound of the actual DNS query on cache miss.

The result, illustrated in Fig. 10, shows a long-tail pattern in both cases. When cache hit occurs, the query latency is smaller than 2ms in 58.2% of the cases. But there are also 2.8% of the cases where the latency is longer than 50ms. In the case of cache miss, the latency grows, and the tail expands further since the DNS resolver needs to query the authoritative name server. The result shows that 20.4% of the queries experience latency over 200ms, which severely delays the connection setup latency.

Anycast routing. Cloudflare is a leading CDN service provider that is built on anycast. It provides service via 152 data centers around the world. Its public DNS servers are exposed via an anycast IP address 1.1.1.1 and are hosted globally. We use 500 RIPE Atlas probes as vantage points (VPs) to make DNS requests to public DNS servers. The requested domain is hosted on an authoritative name server owned by ourselves. The public DNS servers act as resolvers and forward the query to our name server for DNS records, with a unicast IP address as the source address. In this way, the unicast address of the DNS resolver is discovered.

With the unicast IP address of each anycast DNS resolver, it is possible to check if the anycast routing directs a VP to the DNS resolver with the minimal RTT. We use RIPE's ping utility to measure the RTT between each VP and DNS resolver. $latency_{anycast}$ is the measured RTT from a VP to an anycast DNS resolver's unicast representative, and $latency_{min}$ is the minimal RTT from a VP to all of the DNS resolvers. Among the results, 82.3% of the VPs have witnessed that $latency_{anycast}$ equals $latency_{min}$. It indicates that anycast routing already provides good performance in most cases. For the remaining VPs where the anycast routing does not lead to the optimal replica, we show the distribution of the RTTs in Fig. 11. Analyzing the additional latency $latency_{extra} = latency_{anycast} - latency_{min}$, we find that in 31.6% of the cases, $latency_{extra}$ is smaller than 2ms, and in 15.8% of the cases, $latency_{extra}$ is larger than 20ms. On average, anycast routing causes an additional latency of 12.71ms compared with the scenario where the client is connected to the nearest DNS resolver.

Cloudflare deploys its frontend and backend DNS resolvers at the same data center [47]. Therefore, it is reasonable to abstract the complexity of frontend and backend DNS resolvers as a single entity at the scale of data centers. Our anycast routing measurement method works based on

7. https://aws.amazon.com/route53/

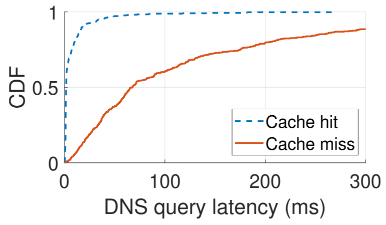


Fig. 10: DNS query latency in cache hit and cache miss cases.

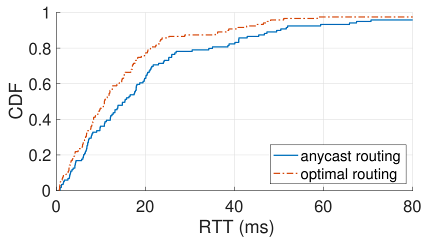


Fig. 11: End-to-end latency by anycast routing and the optimal routing.

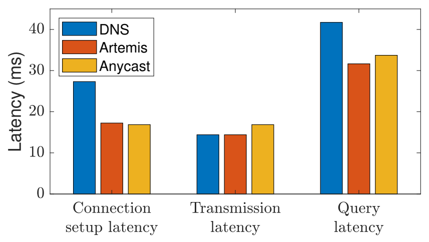


Fig. 12: Comparison of latency with DNS, Artemis, and anycast.

System	Data path	Delay configuration	Ratio
DNS	$C \rightarrow DNS$	cache hit in Fig. 10	83.5%
		cache miss in Fig. 10	16.5%
	$C \rightarrow S_1$	optimal routing in Fig. 11	100%
Anycast	$C \rightarrow S_1$	anycast routing in Fig. 11	100%
Artemis	$C \rightarrow SD_1$	anycast routing in Fig. 11	100%
	$SD_1 \rightarrow S_2$	ignored	82.3%
	$SD_1 \rightarrow SD_2 \rightarrow S_3$	estimated by distance	17.7%
	$C \rightarrow S_2/S_3$	optimal routing in Fig. 11	100%

TABLE 3: Network parameters of the simulation setup.

this assumption. It is more reliable than existing CHAOS-record-based solutions because the CHAOS record may cause ambiguities [48]. However, this method may not work on the DNS deployments where the frontend DNS resolvers and the invoked backend resolvers reside in different data centers. In this case, a latency-based solution [49], which considers the constraint caused by the speed of light, shall be a better solution.

5.2.2 Test cases

We simulate empirical end-to-end network latencies according to the patterns shown in previous latency measurements. We consider the scenario where replica servers are deployed in all available data centers. Three test cases are created to represent service discovery over DNS, anycast, and Artemis, respectively. The setup of the simulation is described in Table 3.

- **DNS**. The direct connection shown in Fig. 7 is used. The link delay between the client and the server is simulated based on the connection delay in optimal routing, as shown in Fig. 11. The DNS query delay is simulated based on the measurement result shown in Fig. 10. According to [50], we set the DNS cache hit ratio to 83.5% in the simulation.
- **Anycast**. The direct connection shown in Fig. 7 is used. The link delay between the client and the server is simulated based on the connection delay in anycast routing, as shown in Fig. 11.
- **Artemis**. The connection over Artemis shown in Fig. 7 is used. The link delay between the client and the server is negligible because they are inside the same data center. The link delay between two Service Dispatchers is calculated based on the geographical distance: $latency_{overlay} = \frac{2.3 \times distance}{c}$. c is the speed of light, and 2.3 is the median factor that network latency is compared with the speed of light [51]. The probability of the two scenarios in Artemis is the same as the probability that anycast routing happens to select the replica server of the lowest latency, i.e., 82.3% for connection over Artemis (1) and 17.7% for connection over Artemis (2) in Fig. 7.

Connection setup latency. As shown in Fig. 12, the mean value of the connection setup latency in DNS is 27.32ms. In Artemis, this process takes 17.25ms on average. So, *Artemis reduces the connection setup latency by 36.8% compared to DNS*. The reduction in latency is mainly contributed by eliminating of dedicated name resolution packets. The ServiceID is calculated locally on the client-side, and the initial packet is generated based on the target DNS name immediately. On the contrary, DNS requires a network-based query before connecting to the server, which takes time. The only factor that may delay Artemis is the routing of anycast.

How anycast routing affects Artemis? In rare cases, anycast routing may direct the client's initial packet to a distant replica server and make the connection setup latency as high as 200ms. Most anycast-based systems suffer from this problem [23], [26]. In Artemis, our solution to this problem is sacrificing a little bit in the handshake latency but ensuring that a client always selects the nearest replica server. It is a worthy trade-off because handshake latency occurs only once, but a client can always benefit from the reduction in RTT during the lifetime of a connection. According to the simulation result, *Artemis reduces 2.46ms in RTT compared to legacy anycast routing by reselecting a closer server replica*. The side effect is the increment of 0.39ms in the connection setup latency, which is much smaller than the reduction in RTT. Some solutions have been proposed to improve anycast routing. For example, [52] adds geographic hints to BGP advertisements and obtains encouraging results. Artemis can benefit from them.

Query latency. The query latency is composed of the connection setup latency and one RTT between the client and the replica server. Since both DNS and Artemis find the optimal replica server for a client, the RTT is the same for both. Anycast obtains a longer RTT because anycast routing does not always find the nearest replica server. According to the simulation result, the client completes the query within 41.72ms, 33.72ms, and 31.65ms for DNS, anycast, and Artemis, respectively. Artemis achieves the lowest latency by performing name resolution at the same time of, instead of before, establishing transport layer connection and making up the routing issue in anycast. *Compared with anycast, Artemis is load-aware, and the query latency is 6.1% lower. Compared with DNS, Artemis reduces the query latency*

Test case	# of Data centers	# of Continents
Global Large	11	5
Global Small	5	5
Regional Europe	5	1
Regional US	5	1

TABLE 4: Deployment parameters of the test cases

by 24.13%.

5.3 Real-world Deployment

In this section, we go beyond simulation and deploy Artemis on the real Internet to further evaluate its performance.

Experiment setup. We deploy Artemis infrastructure on Google Cloud as described in Section 4.3. The clients are deployed within 15 AWS regions across 5 continents[8]. The latency from each data center to each client region is measured beforehand. It takes 11 $\times$ 15 = 165 rows in the *LatencyMeasurement* table to store the latency measurement.

Test cases. As studied in [20], the performance of an anycast-based system is affected by the number and the geo-location of the replica servers. As shown in Table 4, we design four test cases to evaluate how Artemis performs under different deployment options. In the global large case, we deploy clients over 15 data centers and servers over 11 data centers. The scale is not as large as global commercial deployment of hundreds of data centers. We use the name "global large" to distinguish it from other test cases of smaller scales. Besides, as analyzed in the next paragraph, the experiment result shows that such a deployment scale already exhibits a better system performance than a small-scale deployment.

Experiment result. Fig. 13 compares handshake latency and transmission latency in each test case. The results are summarized in Table 5. We divide the test cases into 3 categories to further analyze the result.

- Global Large. In this case, anycast routing directs most clients to their nearby servers. From Fig. 13, we find that most clients have the same handshake latency and transport latency between anycast and DNS (excluding DNS query latency) except client 9. The result aligns with the conclusion in [20] that 11 anycast nodes are sufficient to serve the world with low latency. More anycast nodes do not contribute much to reducing the latency. Regarding latency, Artemis performs similarly to anycast because anycast is already optimal in this case. DNS has a much higher delay in both connection setup latency and query latency because of the additional DNS query latency. We notice that Artemis is 2.3ms slower than anycast in handshake latency. The additional delay is mainly spent on MySQL query and packet forwarding in the Service Dispatchers. Even with additional processing delays, benefiting from the fix of routing problems in anycast, the query latency of Artemis is only 0.2ms larger than anycast. Focusing on client 9, it shows that Artemis spends additional time on the handshake process to fix the routing problem of anycast and finally achieves a query latency lower than anycast.
- Global Small. In this case, anycast routing performs poorly. The connection setup latency and the query latency of anycast are much larger than DNS because anycast routing directs most clients to distant servers. Artemis also suffers from this issue when anycast routing performs poorly, causing the handshake latency to be as high as anycast. However, thanks to the overlay routing of Artemis, the routing problem of anycast is fixed after the handshake procedure, resulting in a greatly reduced query latency. Compared with anycast, Artemis reduces the query latency by 38.2ms with the side-effect of increasing the connection setup latency by only 1.4ms.
- Regional. In this case, the performance of Artemis, DNS, and anycast are close to each other because anycast routing directs most clients to a nearby server. The performance of the DNS-based solution is slightly worse than Artemis and anycast because of the additional latency introduced by the DNS query. Artemis tends to have higher handshake latency but lower transmission latency than anycast because Artemis always redirects clients to the server with the lowest end-to-end latency, although the one chosen by anycast is only a few milliseconds higher than the closest one. The benefit of this behavior is the reduction of end-to-end latency, which helps the client save a few milliseconds on the transmission delay of all subsequent packets.

In summary, Artemis achieves low-latency naming and routing over a global deployment scale. With the deployment of 11 zones across 5 continents, Artemis reduces the connection setup latency and the query latency by $(38.3-23.2)/38.3 = 39.4\%$ and $(56.5-41.0)/56.5 = 27.4\%$, respectively, comparing with DNS. Artemis also suffers from the high latency caused by anycast routing when anycast routing performs badly. Still, Artemis can fix the routing problem and redirect the client to its nearest replica server. With the deployment of 5 zones across 5 continents, Artemis reduces the query latency by 25.2% with the side-effect of increasing the connection setup latency by 1.8%, compared with anycast.

5.3.1 *Redundancy Routing*

In this subsection, we analyze how Artemis performs under different settings of redundancy parameters. We use the experiment setup the same as the Global Large test case, i.e., replica servers are deployed in 11 regions across 5 continents.

Test cases. We design four test cases and tune the duplication parameter X, which is defined in Section 3.4.3, from 1 to 4 in each test case. When X is set to 1, the Service Dispatcher only forwards the client's request to its optimal data center based on the end-user mapping result. When X is set to 4, the Service Dispatcher clones the client's handshake request into 4 copies, which are sent to the top 4 data centers based on the end-user mapping result.

8. The clients' regions are Japan, Korea, India, Singapore, Australia, Canada, Germany, Sweden, Ireland, United Kingdom, France, N. Virginia, Ohio, N. California, and Oregon. The client index in Fig. 13 follows the same order as the above-listed regions.

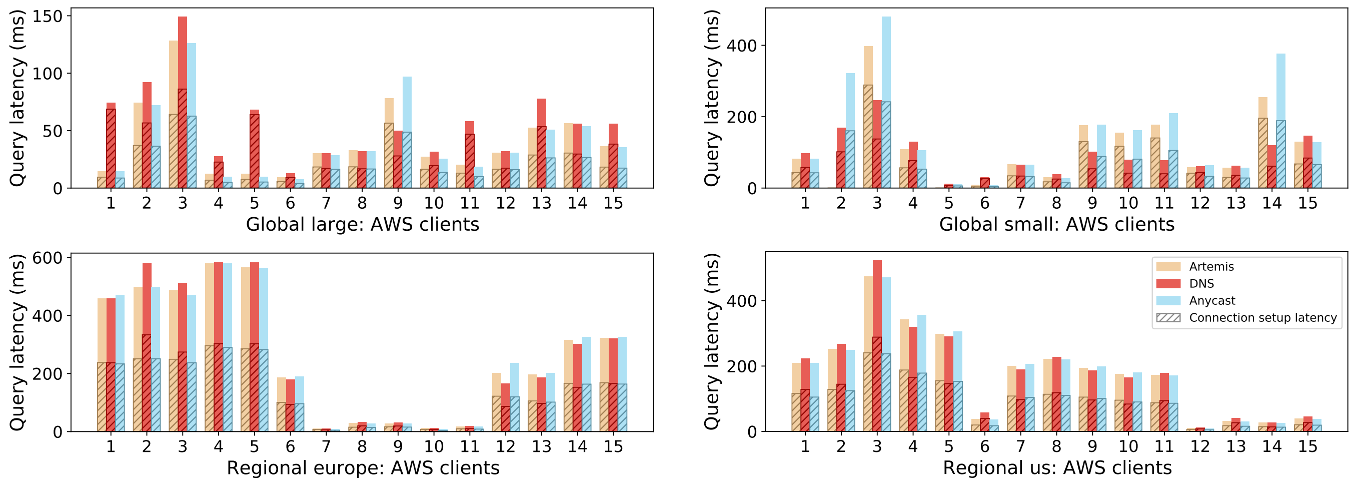


Fig. 13: Comparison of transport-layer performance between Artemis, DNS, and anycast under different test cases. The entire bar represents the query latency. The shadow part of the bar represents the connection setup latency, which is a composition of the query latency.

Test cases	Global Large			Global Small			Regional Europe			Regional US		
Service discovery method	Artemis	DNS	Anycast	Artemis	DNS	Anycast	Artemis	DNS	Anycast	Artemis	DNS	Anycast
Mean connection setup latency (ms)	23.2	38.3	20.9	77.8	55.1	76.4	135.9	140.6	132.4	94.6	98.7	90.8
Mean query latency (ms)	41.0	56.5	40.8	113.4	95.8	151.6	260.9	265.5	263.7	179.0	183.7	180.2

TABLE 5: The statics of comparing Artemis with DNS and anycast.

Test Cases	Optimal Routing Ratio	Anycast Additional Latency (ms)
Global Large	20.0%	1.78
Global Small	20.0%	41.08
Regional Europe	13.3%	6.00
Regional US	6.7%	4.33

TABLE 6: Probability of anycast's sub-optimal routing and the additional latency caused by the sub-optimal routing under different server deployment options.

Parameter X	Data Center Candidates			
	1st	2nd	3rd	4th
1	100%	/	/	/
2	100%	0%	/	/
3	93.3%	6.7%	0%	/
4	93.3%	6.7%	0%	0%

TABLE 7: The percentages of clients that reach different data centers

Experiment result. Table 7 shows the percentile of requests assigned to different data centers. Although the top X data centers are used, most connections are still established between the client and its optimal data center. This phenomenon validates the efficiency of end-user mapping as the optimal one is still selected when candidates compete. We find that end-user mapping already performs well: the chosen data center is the optimal one in over 90% of the cases. Note that our evaluation method is not the same as how a real Artemis runs: for the simplicity of running, we update end-user mapping at the beginning of the experiment. This is why the optimal server is correctly selected by end-user mapping for almost all of the clients. However, in a real deployment of Artemis, the end-user mapping is difficult if not impossible to be up-to-date at the time of client query. Refreshing the end-user mapping usually takes time because updating a huge number of rows in the routing databases is a significant burden to the system.

Our real-world evaluation has demonstrated a promising reduction in various latency metrics from Artemis, comparing with both DNS and anycast. In the future, we plan to deploy real Internet services over Artemis and evaluate with world-wide users, inspecting new challenges brought by the huge number of users to Artemis, especially on its routing infrastructure.

6 Discussion

In this section, we discuss topics not covered in the previous sections, i.e., a comparison between Artemis and DNS in Section 6.1, the feasibility of deploying Artemis in cloud services in Section 6.3, and the security level of Artemis in Section 6.4.

6.1 Artemis v.s. DNS

Artemis is a private system for service discovery, and DNS is a public global name service. Although they are different application targets, both can be used for name resolution and service discovery. In this subsection, we compare Artemis with DNS from these two aspects, and the summary is shown in Table 8.

Category	Functionality	DNS	Artemis
Name resolution	Query message	Dedicated packets	Integrated with QUIC handshake packets
	Cache invalidation	By client	By client and server
	Record update delay	up to 30 minutes [53]	No significant delay
	Security	Encrypted (optional)	Encrypted (mandatory)
	Name space	Global unique	Localized
End-user mapping	Custom policy	Supported	Supported
	Query delay	up to seconds [50]	No significant delay
	Propagation delay	up to 30 minutes	No significant delay

TABLE 8: A comparison between DNS and Artemis in terms of name resolution and end-user mapping

Name resolution is the basic functionality of DNS. With the help of dedicated DNS query and response packets, DNS achieves the translation from a DNS name to an IP address. This process causes name resolution delay as the transmission of packets takes time. In Artemis, the name resolution delay is eliminated because it is completed along with the transport layer handshake process without relying on additional packets. A DNS client knows where to send DNS queries through specific protocols, e.g., dynamic host configuration protocol (DHCP), or a manual configuration, e.g., 8.8.8.8. Similarly, an Artemis client needs to know the service prefix beforehand by obtaining an SDK from the service provider. To save the name resolution delay and reduce server load, DNS employs the cache mechanism, where the DNS query result is cached on the client, and new queries to the same DNS name will never be sent until its TTL expires. The change in DNS records fails to take effect on the client immediately because there is no way for the server to invalidate the client-side cache. In Artemis, the client-side cache can be invalidated by not only the TTL but also the server. Whenever a transport layer handshake with the cached name resolution result fails, the client immediately invalidates the cache and rolls back to the normal name resolution process in Artemis. Regarding security, in Artemis, the name resolution information embedded in the handshake packet is also protected by the transport layer encryption. DNS provides a similar security level via the use of DNS Security Extensions (DNSSEC).

DNS may also be used for service discovery by returning different responses to the same DNS name [8], [14]. With end-user mapping [14], the authoritative name server's response is based on the client's location and server's load, thus achieving latency-oriented routing and load balancing. From this point on, both DNS and Artemis can direct a client to the best server replica based on custom policies. However, as mentioned above, DNS suffers from additional query latency and inevitable information propagation delay, which are the primary strength of Artemis compared with DNS.

Although being of similar functionality, Artemis is not meant to replace DNS. Instead, it provides an alternative naming solution that could coexist with DNS. Artemis is more suitable for the scenario where replica servers are deployed globally. One potential scenario is the cloud-based global services. Cloud providers register anycast network prefixes, distribute Artemis names to the customers and support the deployment of the Artemis infrastructure. Customers deploy replica servers directly in the selected cloud's data centers. End-users can obtain low-latency services by applying the ServiceID mapping patch provided by the cloud.

6.2 Artemis v.s. Anycast

Anycast routing is an effective service discovery solution because of its efficiency in routing packets to the nearest replica server and its native support in the Internet routing. Despite its benefits, its limitations are obvious, for example, lack of application layer control [8], difficulty in management [5], and possible of poor end-to-end latency [52]. To address these issues, Artemis relies on an overlay network to enhance anycast routing.

Application layer control. Application layer control requires the packet routing to be instructed not only by network metrics, e.g., end-to-end latency, but also by application-related metrics, e.g., server load. Artemis supports application layer control by allowing a replica server to be added to or to be removed from the routing infrastructure dynamically. For example, an overloaded server can remove itself from the routing database to avoid new incoming connections and can add itself back when the incoming traffic has reduced to a reasonable level. In case a packet arrives at an anycast zone without any replica server for the specific application, Artemis automatically redirects it to a nearby zone with a proper serving capability. In summary, with the help of the overlay network, Artemis supports more packet routing criteria and is more robust to the route change, comparing with IP anycast.

Easy management. In general, it is difficult to deploy a service on top of IP anycast globally because the network administrator must obtain an address block of adequate size (e.g., /24) and advertise it via BGP to its upstream ISP. Similar to [5], Artemis serves as an anycast-based infrastructure to help the network administrator address these deployment challenges. By deploying over Artemis, developers can focus on building a server running on top of a unicast IP address without worrying about the difficulty in managing an anycast network.

Low end-to-end latency. Ideally, IP anycast routes packets to the nearest of a group of hosts according to the minimal-hop rule. However, the reached host might be a bit far away from the packet sender, although there are closer options. A measurement on the DNS D-root servers shows that anycast routing finds a host further than 1,000km away from the nearest host in about 10% of the cases [52]. To avoid the unnecessary latency caused by such unexpected routing, Artemis redirects packets back to their nearest anycast zones over the overlay network. In our evaluation setup as shown in Section 5.2, Artemis, on average, saves the end-to-end latency by 5.0% from anycast routing.

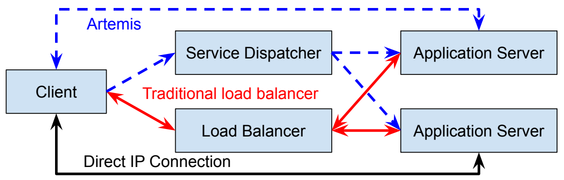


Fig. 14: Compare Artemis with traditional load balancers

6.3 Deployment Requirements

In this subsection, we discuss the feasibility of deploying Artemis in cloud services.

Easy integration into the existing network infrastructure. Artemis can be integrated in a cloud service as an alternative to its existing load balancer. As shown in Fig. 14, the client communicates with the server via the latter's public IP address in the classic client-server model. Adding the load balancer as a proxy between the client and the server allows the cloud service to distribute client requests evenly among application servers. Usually, the load balancer needs to keep records of the live connections to ensure the binding between a client and a server [18], [54]. Artemis works the same way as legacy load balancers, with the difference that only the handshake packets go through its overlay routing, with Service Addresses, while the other packets are transmitted directly between the client and the server, with the servers' unicast IP address.

Regarding this, deploying Artemis in a cloud service requires 1) replacing the implementation of legacy load balancers with Service Dispatchers and 2) adding additional NICs to the application server.

Changes in application servers. As mentioned in Section 4.2, Artemis does not require any change on the QUIC implementation, as long that the *preferred_address* feature is appropriately implemented. In the application code, there needs to be a minor change to make QUIC listen on two UDP ports simultaneously, one on the anycast address and the other on the unicast address. In practice, it only requires a few lines of code dealing with multiple sockets.

Delivery to clients. The client needs to obtain an SDK, which implements the name resolution function as mentioned in Section 4.2, from the cloud service the first time using Artemis. Also, the application code should be modified to invoke Artemis, instead of DNS, for name resolution as mentioned in Section 4.2, which requires only a single line of code change. An SDK is bound with an Artemis deployment because the deployment's Service Prefix, i.e., anycast IP prefix, is hard coded into the SDK. In practice, the application service provider may build an Artemis variation of its client program to allow its clients to use Artemis, instead of DNS, for name resolution.

6.4 Security Concerns

Artemis is designed carefully with security in consideration. In this subsection, we study representative attack cases, which demonstrate how Artemis is resilient to malicious attackers.

Distributed denial-of-service (DDoS) attack. With DDoS, the attacker aims to overload a server with a large number of queries from a bunch of clients. Artemis protects both itself and its hosted application servers from such attacks. For Artemis, its routing infrastructure, i.e., Service Dispatchers, is stateless, i.e., does not maintain connection state and end-user mapping records, making a single router able to handle a large number of connections and more routers able to work as duplicates for sharing loads. For application servers, like other load balancers [55], Artemis supports temporarily removing a heavily loaded server from the overlay routing table (Section 3.4) to avoid overloading. In this way, the requests exceeding a server's capability of handling will be redirected to nearby servers.

Man in the middle (MITM) attack. With MITM, the attacker secretly sniffs and possibly tampers messages between two communicators by pretending to be the other participant in the middle. The solution to this attack is encrypting the messages and verifying the other end-host's identity via a trusted certificate authority [56], which is supported by QUIC [43], i.e., TLS 1.3. As Artemis is domain-oriented and the applications' servers are exposed by Artemis names instead of IP addresses, the client can always actively check the server's identity to avoid MITM attacks.

Name resolution hijacking. In Artemis, the risk exists that the Service Dispatchers may maliciously route packets to servers who do not own the target Artemis name, i.e., hijacking a name. The same problem exists in DNS, where an LDNS or a public DNS server can easily respond with a malicious DNS name. The same as DNS, hijacking is not avoidable but detectable in Artemis. In Artemis, all application servers must own an X.509 certificate [43] signed by a trusted authority to prove their ownership of the names. In this way, the client can verify that it is connected to a real server, instead of a pretended one, through the online certificate status protocol (OCSP). The OCSP lookup latency can be further eliminated by using OCSP stamping.

In summary, benefiting from the QUIC and TLS 1.3's security model, Artemis is resilient to representative attacks. And the scalable design in Artemis makes the system robust on heavy loads.

7 Conclusion and Future Work

In this paper, we propose Artemis, a low-latency naming and routing system that reduces the connection setup latency by eliminating the name resolution latency. With an additional layer composed of Service Dispatchers, Artemis can intervene in the handshake process and achieve optimal server selection for the clients based on customizable policies. The simulation shows that Artemis reduces the average connection setup latency by 36.8%, compared with the state-of-the-art DNS solution. Artemis represents an exciting new opportunity of low latency service discovery for globally deployed Internet services.

The evaluation over commercial clouds shows that Artemis reduces both the connection setup and end-to-end latency compared with DNS and anycast, whether deployed regionally or globally. The latency would further decrease when Artemis becomes available in more data centers. In our deployment with 11 globally-scaled data centers, Artemis reduces the connection setup latency and the transmit latency by 39.4% and 27.4%, respectively, compared with DNS.

The design of Artemis is coupled with the transport layer protocol because it modifies the handshake packets for name resolution. In this paper, we take advantage of QUIC's support in connection migration and packet customization and present a prototype of Artemis. In the future, we plan to implement Artemis over other transport layers protocols that support late binding.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61971145) and the HUAWEI research collaboration YBN2019125184. This work was partially funded by Academy of Finland (No. 317432).

References

[1] "Latency is Everywhere and it Costs You Sales - How to Crush it," Jul. 2009. [Online]. Available: http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it

[2] E. Schurman and J. Brutlag, "Performance related changes and their user impact," in *Proc. O'Reilly Velocity Web Performance Operations Conf.*, 2009.

[3] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, "Comparing DNS resolvers in the wild," in *Proc. ACM Int. Measurement Conf.*, 2010, pp. 15–21.

[4] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with WProf," in *Proc. USENIX Symp. Netw. Syst. Design Implementation*, 2013, pp. 473–486.

[5] H. Ballani and P. Francis, "Towards a global IP anycast service," in *Proc. ACM SIGCOMM*, 2005, pp. 301–312.

[6] M. Prince, "A Brief Primer on Anycast," Oct. 2011. [Online]. Available: https://blog.cloudflare.com/a-brief-anycast-primer

[7] F. Chen, R. K. Sitaraman, and M. Torres, "End-User Mapping: Next Generation Request Routing for Content Delivery," in *Proc. ACM SIGCOMM*, 2015, pp. 167–181.

[8] A. Flavel, P. Mani, D. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev, "FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs," in *Proc. USENIX Symp. Netw. Syst. Design Implementation*, 2015, pp. 381–394.

[9] F. Wohlfart, N. Chatzis, C. Dabanoglu, G. Carle, and W. Willinger, "Leveraging interconnections for performance: the serving infrastructure of a large CDN," in *Proc. ACM SIGCOMM*, 2018, pp. 206–220.

[10] X. Li, B. Liu, Y. Chen, Y. Xiao, J. Tang, and X. Wang, "Artemis: A Practical Low-latency Naming and Routing System," in *Proc. ACM Int. Conf. Parallel Processing*, 2019, pp. 60:1–60:10.

[11] M. Thomson and J. Iyengar, "QUIC: A UDP-Based Multiplexed and Secure Transport," 2021. [Online]. Available: https://doi.org/10.17487/rfc9000

[12] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory TCP: connection migration for service continuity in the Internet," in *Proc. IEEE Int. Conf. Distributed Comp. Syst.*, 2002, pp. 469–470.

[13] B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, *The Site Reliability Workbook: Practical Ways to Implement SRE*. O'Reilly Media, Inc., 2018.

[14] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-performance Internet Applications," in *Proc. ACM SIGOPS*, 2010, pp. 2–19.

[15] Y.-D. Song, A. Mahanti, and S. C. Ravichandran, "Understanding Evolution and Adoption of Top Level Domains and DNSSEC," in *Proc. IEEE Int. Symp. Measurements Netw.*, 2019, pp. 1–6.

[16] T. Jarassriwilai, T. Dauber, N. Brownlee, and A. Mahanti, "Understanding evolution and adoption of Top-level Domain names," in *Proc. IEEE Local Comp. Netw. Conf. Workshops*, 2015, pp. 687–694.

[17] C. Contavalli, W. van der Gaast, D. C. Lawrence, and W. Kumari, "Client Subnet in DNS Queries," 2016. [Online]. Available: https://doi.org/10.17487/rfc7871

[18] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: a fast and reliable software network load balancer," in *Proc. USENIX Symp. Netw. Syst. Design Implementation*, 2016, pp. 523–535.

[19] Y. Rekhter and K. Lougheed, "Border Gateway Protocol (BGP)," 1989. [Online]. Available: https://doi.org/10.17487/rfc1105

[20] R. de Oliveira Schmidt, J. Heidemann, and J. H. Kuipers, "Anycast Latency: How Many Sites Are Enough?" in *Proc. Springer Int. Conf. Passive Active Netw. Measurement*, 2017, pp. 188–200.

[21] G. C. Moura, R. d. O. Schmidt, J. Heidemann, W. B. de Vries, M. Muller, L. Wei, and C. Hesselman, "Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event," in *Proc. ACM Int. Measurement Conf.*, 2016, pp. 255–270.

[22] S. Sarat, V. Pappas, and A. Terzis, "On the use of anycast in DNS," in *Proc. IEEE Int. Conf. Comp. Communication Netw.*, 2006, pp. 71–78.

[23] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in *Proc. Int. Measurement Conf.*, 2015, pp. 531–537.

[24] D. Cicalese, J. Augé, D. Joumblatt, T. Friedman, and D. Rossi, "Characterizing IPv4 anycast adoption and deployment," in *Proc. ACM Conf. Emerging Netw. Experiments Technologies*, 2015, pp. 1–13.

[25] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of Google's serving infrastructure," in *Proc. ACM Int. Measurement Conf.*, 2013, pp. 313–326.

[26] L. Wei and J. Heidemann, "Does Anycast Hang Up on You (UDP and TCP)?" *IEEE Trans. Netw. Service Management*, vol. 15, no. 2, pp. 707–717, 2018.

[27] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, and Z.-L. Zhang, "A tale of three CDNs: An active measurement study of Hulu and its CDNs," in *Proc. IEEE INFOCOM Workshops*, 2012, pp. 7–12.

[28] L. Wang, V. Pai, and L. Peterson, "The effectiveness of request redirection on CDN robustness," in *Proc. ACM SIGOPS*, 2003, pp. 345–360.

[29] R.-H. Huang, B.-j. Chang, Y.-L. Tsai, and Y.-H. Liang, "Mobile Edge Computing-Based Vehicular Cloud of Cooperative Adaptive Driving for Platooning Autonomous Self Driving," in *Proc. IEEE Int. Symp. Cloud Service Comp.*, 2017, pp. 32–39.

[30] Y. Zhang, J. Jiang, K. Xu, X. Nie, M. J. Reed, H. Wang, G. Yao, M. Zhang, and K. Chen, "BDS: a centralized near-optimal overlay network for inter-datacenter data replication," in *Proc. ACM EuroSys*, 2018, pp. 1–14.

[31] P. Medagliani, S. Paris, J. Leguay, L. Maggi, C. Xue, and H. Zhou, "Overlay routing for fast video transfers in CDN," in *Proc. IFIP/IEEE Symp. Integrated Netw. Service Management*, 2017, pp. 531–536.

[32] G. Calvigioni, R. Aparicio-Pardo, L. Sassatelli, J. Leguay, P. Medagliani, and S. Paris, "Quality of Experience-based Routing of Video Traffic for Overlay and ISP Networks," in *Proc. IEEE INFOCOM*, 2018, pp. 935–943.

[33] D. Zhuo, K. Zhang, Y. Zhu, H. H. Liu, M. Rockett, A. Krishnamurthy, and T. Anderson, "Slim: OS Kernel Support for a Low-Overhead Container Overlay Network," in *Proc. USENIX Symp. Netw. Syst. Design Implementation*, 2019, pp. 331–344.

[34] C. Fang, H. Liu, M. Miao, J. Ye, L. Wang, W. Zhang, D. Kang, B. Lyv, P. Cheng, and J. Chen, "VTrace: Automatic Diagnostic System for Persistent Packet Loss in Cloud-Scale Overlay Network," in *Proc. ACM SIGCOMM*, 2020, pp. 31–43.

[35] T. Hansen and E. Donald E., "US Secure Hash Algorithms (SHA and HMAC-SHA)," 2006. [Online]. Available: https://doi.org/10.17487/rfc4634

[36] I. A. Board, "IAB Technical Comment on the Unique DNS Root," 2000. [Online]. Available: https://doi.org/10.17487/rfc2826

[37] T. Lemon and R. Droms, "Special-Use Domain Names Problem Statement," 2017. [Online]. Available: https://doi.org/10.17487/rfc8244

[38] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," 2018. [Online]. Available: https://doi.org/10.17487/rfc8446

[39] "IPv6 – Google." [Online]. Available: https://www.google.com/intl/en/ipv6/statistics.html

[40] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle, "Scanning the IPv6 Internet: Towards a Comprehensive Hitlist," *arXiv:1607.05179 [cs]*, 2016.

[41] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *Proc. USENIX Symp. Netw. Syst. Design Implementation*, 2013, pp. 605–620.

[42] K. Park, V. S. Pai, L. Peterson, and Z. Wang, "CoDNS: improving DNS performance and reliability via cooperative lookups," in *Proc. USENIX OSDI*, 2004, pp. 199–214.

[43] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, “How Secure and Quick is QUIC? Provable Security and Performance Analyses,” in *Proc. IEEE Symp. Security Privacy*, 2015, pp. 214–231.

[44] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, “The design and implementation of open vSwitch,” in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, 2015, pp. 117–130.

[45] P. Jurkiewicz, G. Rzym, and P. Boryło, “Flow length and size distributions in campus Internet traffic,” *Elsevier Computer Communications*, vol. 167, pp. 15–30, 2021.

[46] V. Bajpai and J. Schönwälder, “A Survey on Internet Performance Measurement Platforms and Related Standardization Efforts,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1313–1341, 2015.

[47] J. L. Martin, “DNS resolver 1.1.1.1 from Cloudflare,” Tech. Rep., 2018. [Online]. Available: https://conference.apnic.net/46/assets/files/APNC402/DNS-resolver-1.1.1.1-from-Cloudflare.pdf

[48] X. Fan, J. Heidemann, and R. Govindan, “Evaluating anycast in the domain name system,” in *Proc. IEEE INFOCOM*, 2013, pp. 1681–1689.

[49] D. Cicalese, D. Joumblatt, D. Rossi, M.-O. Buob, J. Augé, and T. Friedman, “A fistful of pings: Accurate and lightweight anycast enumeration and geolocation,” in *Proc. IEEE INFOCOM*, 2015, pp. 2776–2784.

[50] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, “DNS performance and the effectiveness of caching,” *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 589–603, 2002.

[51] A. Singla, B. Chandrasekaran, P. B. Godfrey, and B. Maggs, “The Internet at the Speed of Light,” in *Proc. ACM Workshop Hot Topics Netw.*, 2014, pp. 1–7.

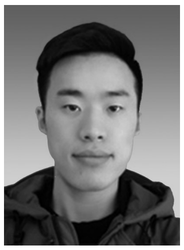
[52] Z. Li, D. Levin, N. Spring, and B. Bhattacharjee, “Internet anycast: performance, problems, & potential,” in *Proc. ACM SIGCOMM*, 2018, pp. 59–73.

[53] Z. Gao and A. Venkataramani, “Measuring Update Performance and Consistency Anomalies in Managed DNS Services,” in *Proc. IEEE INFOCOM*, 2019, pp. 2206–2214.

[54] J. Varia and S. Mathew, “Overview of Amazon Web Services,” 2013. [Online]. Available: http://cabibbo.dia.uniroma3.it/asw-2014-2015/altrui/AWS_Overview.pdf

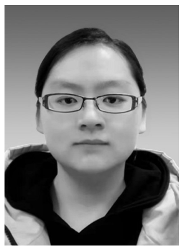
[55] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, “SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs,” in *Proc. ACM SIGCOMM*, 2017, pp. 15–28.

[56] D. Cooper, S. Farrell, S. Boeyen, R. Housley, and T. Polk, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” 2008. [Online]. Available: https://doi.org/10.17487/rfc5280



Xuebing Li received the MSc and BSc degrees in computer science from Fudan University. He is currently a PhD student at Aalto University. His main research interests include network measurements, network protocols, and Internet architectures.

Yang Chen (Senior Member, IEEE) is an Associate Professor within the School of Computer Science at Fudan University, China. He leads the Mobile Systems and Networking (MSN) group since 2014. Before joining Fudan, he was a postdoctoral associate at the Department of Computer Science, Duke University, USA, where he served as Senior Personnel in the NSF MobilityFirst project. From September 2009 to April 2011, he has been a research associate and the deputy head of Computer Networks Group, Institute of Computer Science, University of Göttingen, Germany. He received his B.S. and Ph.D. degrees from Department of Electronic Engineering, Tsinghua University in 2004 and 2009, respectively. He visited Stanford University (in 2007) and Microsoft Research Asia (2006-2008) as a visiting student. His research interests include online social networks, Internet architecture and mobile computing. He is serving as an Editorial Board Member of the Transactions on Emerging Telecommunications Technologies (ETT) and an Associate Editor of Computer Communications. He served as an OC / TPC Member for many international conferences, including SOSP, SIGCOMM, WWW, IJCAI, AAAI, IWQoS, ICCCN, GLOBECOM and ICC. He published more than 50 referred papers in international journals and conferences, including TPDS, TDSC, TMC, TKDE, TSC, TNSM, TCSS, IEEE Communications Magazine, IEEE Software, Middleware, INFOCOM, ICDE, COSN, CIKM and IWQoS.



Mengying Zhou received her BSc degree in information security from Lanzhou University in 2019. Currently she is a Ph.D. candidate at Fudan University. Her main research interests include network measurement, social networks, and data mining.

Tiancheng Guo received his BSc degree in computer science from Fudan University, China. He is currently working toward the MSc degree at Fudan University, China. His research interests include social computing, network routing, and urban mobility.

Chenhao Wang received his BSc degree in computer science from Fudan University, China. He is currently working toward the Ph.D. degree at Fudan University, China. His main research interests include network measurements, network routing, and machine learning.

Yu Xiao (Member, IEEE) received the doctoral degree in computer science from Aalto University, in 2012. She is currently an associate professor with the Department of Communications and Networking, Aalto University. Her current research interests include mobile crowdsensing, augmented reality, and edge computing.

Junjie Wan received the MS degree in information science from the University of North Carolina at Chapel Hill, USA, in 2017. He is currently a senior engineer of Network Technology Laboratory at Huawei, China. His research interests include network architecture, routing protocol and network service.

Xin Wang (Member, IEEE) received his BSc degree in information theory and MSc degree in communication and electronic systems from Xidian University, China in 1994 and 1997, respectively. He received his PhD degree in computer science from Shizuoka University, Japan, in 2002. Currently professor in Fudan University, Shanghai, China. His main research interests include quality of network service, next generation network architecture, mobile Internet and network coding.