AP-atoms: A High-Accuracy Data-Driven Client Aggregation for Global Load Balancing

Yibo Pi, Sugih Jamin
Univerisity of Michigian
Ann Arbor, MI
{yibo, sugih}@umich.edu

Peter Danzig
Panier Analytics
Menlo Park, CA
pbdanzig@gmail.com

Jacob Shaha
United States Military
Academy
West Point, NY
Jacob.Shaha@usma.edu

ABSTRACT

In Internet mapping, IP address space is divided into a set of client aggregation units, which are the finest-grained units for global load balancing. Choosing the proper level of aggregation is a complex and crucial problem, which determines the total number of aggregation units that a mapping system has to maintain and the accuracy of client redirection. In this paper, using Internet-wide measurements provided by a commercial global load balancing service provider, we show that even for the best existing client aggregation, almost 17% of clients have latency more than 50 ms apart from the average latency of clients in the same aggregation unit. To address this, we propose a data-driven client aggregation, *AP-atoms*, which can tradeoff scalability for accuracy and adapts to changing network conditions. Our experiments show that by using the same scale of client aggregations, AP-atoms can reduce the number of widely dispered clients by almost 2× and the 98-th percentile difference in clients' latencies by almost 100 ms.

1. INTRODUCTION

DNS-based client redirection has been adopted by many CDN provides like Google [25] and Akamai [13]. Since the local DNS nameserver (LDNS) of a client is typically co-located with the client, LDNSs are used as a *proxy* for nearby clients. To make redirection decisions for billions of clients, a mapping system only need to measure the performance from servers to hundreds of thousands of LDNSs. However, a recent study on Akamai's CDN found that LDNSs are not a good proxy for nearly 20% of client demand, where clients use either public resolvers or LDNSs that are remote to the clients [14]. To better map these clients to the closest servers, Akamai rolled out the next-generation mapping system, i.e., *end-user mapping*, which locates clients on the Internet using the subnet[1] of the clients rather than their LDNSs. End-user mapping is made possible by the recent extension to the DNS protocol (EDNS), which allows recursive nameservers to carry the subnet of the client in their DNS queries [9]. To take advantage of EDNS for global load balancing, the mapping system has to estimate the path performance from servers to millions of subnets on the Internet. To maintain up-to-date path performance estimation for millions of subnets is not scalable.

Choosing the proper aggregation of clients is to find a good tradeoff between scalability and accuracy. Using /24 IP blocks as client aggregations may be accurate in redirection, but not scalable in terms of performance estimation. In contrast, aggregating clients by their LDNSs is scalable, but not accurate for remote clients. IP blocks with /20 network prefix have been proposed to be a good tradeoff between scalability and accuracy [14]. Clients can also be aggregated by their geographic locations [27] and BGP routing paths [22]. In this paper, we use Internet-wide measurements provided by a commercial global load balancing service provider to study the performance of existing client aggregations.

We find that even for the best existing aggregation, almost 17% of clients have latency 50 ms apart from other clients. To understand the causes for the widely dispersed clients, we categorize clients based on their difference in latency and study if clients far away in latencies also differ significantly in their IP addresses. We find that the wide dispersal client latencies in existing aggregations are caused by aggregating clients based on attributes other than path performance. To address this, we propose *AP-atoms*, which group clients based directly on their path performance to servers. AP-atoms are data-driven, obtained from the traffic data available to service providers without incurring extra measurement loads. To obtain AP-atoms, we use machine learing algorihtms to identify distinct latency patterns and cluster clients based on these patterns.

Our contributions are summarized as follows:

- To the best of our knowledge, we are the first to conduct a comparative study of the performance of existing client aggregation methods.
- We propose a data-driven aggregation method that can flexibly tradeoff scalablity for accuracy. The

[1]A /24 IP block is used (the lowest byte of the client IP is masked) for anonymity.

data-driven property enables our mechanism to dynamically adapt to changing network conditions.

- We propose to use the modes of RTTs as latency and use machine learning algorithms to identify different latency patterns of clients.

2. EXISTING CLIENT AGGREGATIONS

To motivate this study, we first compare the performance of existing client aggregations. To our best knowledge, these existing aggregation methods have been studied separately [14, 20, 24, 27], but there has been no comparative study of the four methods. We use Internet-wide measurements from a commercial global load balancing service provider for the comparative study. Our dataset includes 30 days of measurements. Each day's data contains about 1.5 billion measurement records, occupying approximately 550 GB. Clients in our dataset cover 86% of countries in the world and are from 4.2 million /24s. Path latencies are measured between these clients and 160 CDN and cloud service providers, including Akamai, Amazon, Google, Level3 and Microsoft Azure, and others. Our dataset does not contain users' network traffic and passwords. Further details about our dataset and data processing techniques will be provided in later sections.

The general principle in client aggregation is to pool together clients that are similar. Existing aggregations define client similarity mainly based on one of four attributes: (1) geographic locations, (2) fix-sized prefixes, (3) BGP routing paths, or (4) LDNSs. Using the first attribute, clients within a given geographic radius are gathered together into *geo-blocks* [27]. Aggregation by the second attribute simply groups clients by the first k bits of their IP addresses. Researchers studying Akamai's end-user mapping suggested */20 prefixes* as a good tradeoff between scalability and accuracy [14]. Aggregation by the third attribute exploits shared BGP routing paths amongst clients. Since clients within the same routable prefix share a portion of their BGP paths, it is reasonable to aggregate clients into routable *BGP prefixes*. Aggregation by the fourth attribute is commonly used by CDNs, where clients using the same LDNS are aggregated into the same aggregation unit [14, 25].

Using our dataset, we simulate the four aggregation methods above as used in real systems. In Internet routing, routers choose the next hop to forward packets using the longest prefix match. Packets sent from a server to a client are routed to the prefix that shares the longest common bits with the client's IP address. The latency along the routes is the latency between the server and the client. We thus use the latency measurements between clients and servers in our dataset to simulate the latency along the routes in Internet routing.

(a) Latency dispersion of clients (b) Impact of dispersion

Figure 1: Illustration of latency dispersion

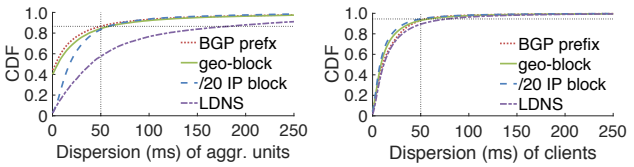
To compare the performance of the four client aggregation methods, we present a metric referred to as *latency dispersion*, which measures the differences in latency between clients in the same aggregation unit. The same concept of dispersion applies to other performane metrics such as bandwidth and packet loss. Since our dataset only includes latency measurements, we study the performance of existing client aggregations in terms of latency.

2.1 Latency Dispersion

As shown in Figure 1(a), an aggregation unit is an aggregation of clients. Given a server, we can have two types of latencies: 1) the latency from *each client* in the aggregation unit to the server and 2) the latency from the aggregation unit *as a whole* to the server.[2] For scalable management of the Internet, client redirections are determined by the latencies of aggregation units to servers [14, 25]. To understand the effects of using latency of aggregation units to perform redirection for all clients inside the unit, we want to know the difference in latencies between clients in the same aggregation unit. We define *latency dispersion* as follows. The *dispersion of a client to a server* is the difference between the latency of the client to the server and the centroid of the aggregation unit to the server, where the centroid is the average latency of clients in the aggregation unit. After we compute the dispersion of each client, we define the *dispersion of an aggregation unit to a server* to be the largest dispersion among all clients in the unit.

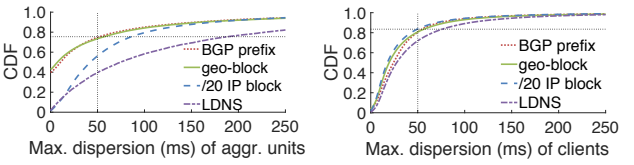
Figure 1(b) shows an example on the impact of dispersion on server selection. Based on latencies, server 1 is a better choice than server 2 for the aggregation unit. However, since the dispersion of the aggregation unit to server 1 is 100 ms, directing all clients in the unit to server 1 will cause some clients to experience 50 ms larger latency than if they had been redirected to server 2. To improve the redirection accuracy for all clients in an aggregation unit, we prefer that the dispersion of the aggregation unit to all servers be small. In the following, we first show the latency dispersion of existing client aggregations to a single server and then

[2]To determine the latency of an aggregation unit, one client in the unit can be selected randomly and its latency is considered as representative of the unit [25].



(a) (b)

Figure 2: Latency dispersion of existing client aggregations to a single server



(a) (b)

Figure 3: Maximum latency dispersion of existing client aggregations to multiple servers

the maximum latency dispersion to all servers.

2.1.1 *Latency Dispersion to a Single Server*

We use the database of routable BGP prefixes from CAIDA [7] for aggregation by BGP routing paths and the IP-to-location database from IP2Location [18] for aggregation by geographic locations. We aggregate IP blocks into geo-blocks with a radius of 200 miles. This results in the 90-th percentile size of geo-blocks equivalent to /20 IP address block size, which incidentally is also the same as the 89-th percentile size of BGP prefixes. We use /20 IP blocks for fix-sized prefixes and aggregate clients by their LDNSs. We compare the latency dispersion of the four aggregation methods above on two granularities: aggregation units and clients.

To calculate latency dispersion, we obtain the latencies from 1.7 million clients to a given server and map clients to the corresponding aggregation units using the longest prefix match for each type of client aggregations. Then, we calculate the dispersion of aggregation units and clients. Figure 2(a) shows the distribution of latency dispersion of *aggregation units* to a given server. BGP prefixes in general have smaller dispersion than other aggregations, but still have a significant percentage (about 14%) of aggregation units with dispersion larger than 50 ms. Following the study of Google's CDN [25], we use 50 ms as a threshold to indicate a significant difference in latency. Looking at the distribution of latency dispersion of *clients* in Figure 2(b), /20 IP blocks have smaller dispersion than other aggregations. About 6% of clients in /20 IP blocks have dispersion larger than 50 ms. Moreover, the distribution of dispersion of clients has a long tail, where the 99-th percentile dispersion is about 150 ms for all aggregations.[3]

2.1.2 *Latency Dispersion to Multiple Servers*

In global load balancing, since clients of an aggregation unit could be directed to one of a set of candidate servers [6], the worst-case performance of clients is determined by the server to which the aggregation unit has the maximum dispersion among all candidate servers. Thus, we want to know the maximum dispersion of aggregation units and of clients to multiple servers. To calculate the maximum dispersion, we use latencies from 2.3 million clients to 10 servers. The dispersion of clients to each server is first calculated. If a client has no measured latency to a server, the dispersion of the client to the server is considered undetermined. Each client then will have dispersions calculated for up to 10 servers. Among these dispersions, the maximum one is used as the maximum dispersion of clients. Recall how we obtained the dispersion of aggregation units to each server from the dispersion of clients. Similarly, we can obtain the maximum dispersion of aggregation units. Figure 3(a) shows the distribution of the maximum dispersion of aggregation units to the 10 servers. BGP prefixes and geo-blocks have smaller dispersion than /20 IP blocks and LDNS, but almost 26% of BGP prefixes and geo-blocks have dispersion larger than 50 ms to at least one server. Figure 3(b) shows that about 17% of clients in /20 IP blocks and geo-blocks have dispersion larger than 50 ms to at least one server.

Comparing Figures 3(a) and 3(b), we can see that even though BGP prefixes and /20 IP blocks have similar latency dispersion of clients, BGP prefixes have 20% more aggregation units with dispersion less than 50 ms than /20 IP blocks do. This is because BGP prefixes and geo-blocks have 40% of aggregation units with zero dispersion, where 98% of these aggregation units are /24s. However, compared to /20 IP blocks, BGP prefixes and geo-blocks both have very large aggregation units, which could easily have large dispersion if clients included are not similar. Indeed, among the aggregation units with dispersion larger than 250 ms in BGP prefixes, 73% are of size at least /19 while only 7% are of size /22 or smaller. Similarly, among aggregation units with dispersion larger than 250 ms in geo-blocks, 80% are of size at least /19 and only 4% are of size /22 or smaller. As the numbers show, aggregating clients by attributes could result in widely dispersed clients, regardless of aggregation unit sizes.

2.2 Causes for Large Dispersion

An aggregation unit has large dispersion when the

[3]In Figure 2(a), BGP prefixes and geo-blocks include a portion of aggregation units with zero dispersion. This is due to $> 50\%$ of aggregation units from BGP prefixes and geo-blocks being of size /24, comprising of only one client each.

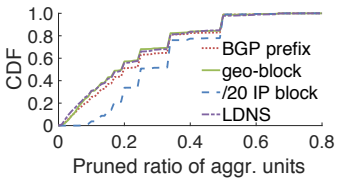


Figure 4: Distribution of the pruned ratio

latencies of a small portion of clients (*outliers*) differ significantly from the latencies of other clients in the same aggregation unit,[4] or when the aggregation unit includes clients that should be in different aggregation units (*over-aggregation*).

2.2.1 Identifying Outliers and Over-aggregation

To study the two causes, we look at the aggregation units with dispersion larger than 50 ms and determine the minimum set of clients that must be pruned to obtain dispersion less than 50 ms. We use a greedy algorithm to minimize the number of clients in the pruned set. The greedy algorithm starts from all the clients in the aggregation unit and first prunes the client that has latency furthest from the average latency of clients. Then, the process is repeated with the rest of the clients until the dispersion of the aggregation unit is less than 50 ms. Then, we calculate the ratio of the number of clients in the pruned set to the number of remaining clients (*pruned ratio*). Figure 4 shows the distribution of pruned ratios for the four aggregation methods. Over 70% of aggregation units of any of the four methods have a pruned ratio less than 0.4, which implies that in at least 70% of aggregation units with large dispersion, a majority of clients have dispersion less than 50 ms. This is the reason Figures 3(a) and 3(b) show that the percentage of aggregation units with the same dispersion is larger than that of clients.

We consider an aggregation unit as containing *outliers* if they have a pruned ratio 0.1 or less. Since a /20 IP block includes at most 16 /24 clients, almost no /20 IP blocks have a pruned ratio less than 0.1. Of the other three aggregation methods, 25% to 31% of their aggregation units have a pruned ratio less than 0.1. Among these aggregation units, over 84% of them are of size at least /18 for BGP prefixes and geo-blocks. We find that among aggregation units of size /18 or larger, the percentages of the ones with a pruned ratio less than 0.1 are 81% for geo-blocks, 85% for BGP-prefixes, and 87% for LDNS.

If we now consider aggregation units with a pruned ratio larger than 0.2 to be due to *over-aggregation*, among aggregation units of size /18 or larger, 10% of geo-blocks, 9% of BGP prefixes and 4% of aggregations by LDNSs with large dispersion are due to over-aggregation.

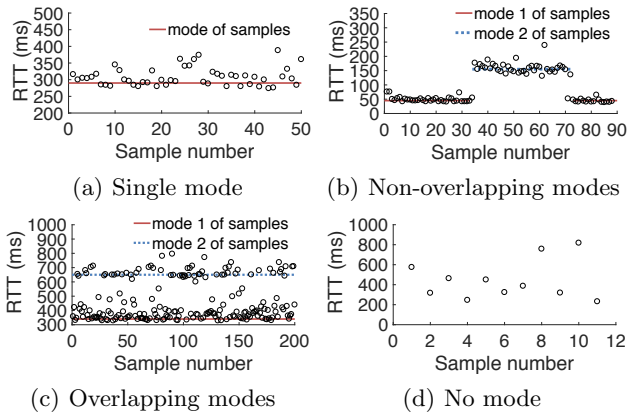
[4]As discussed in Appendix, less than 2% of large dispersion cases are caused by outliers experiencing *atypical* latencies such as caused by network congestion. As has also been asserted in the literature, we speculate that the majority of outliers have limited network resources, such as low access bandwidth [15].

2.2.2 Inflexibility of Existing Aggregations

Existing aggregation methods partition Internet address space by clustering clients similar in certain attributes. The attributes used by the four existing aggregation methods we have studied do not necessarily reflect path performance experienced by clients. Moreover, given an attribute by which clients are aggregated, partitioning of the address space is fixed, not adaptive to changing network conditions. As we have seen, this inflexibility causes the outliers and over-aggregation problems described above. In this section, we further analyze the relationship between inflexible address space partitioning due to the use of arbitrary, non-performance related attributes and the presence of outliers and over-aggregation.

Outliers and over-aggregation are due to clients' having large distances in their latencies (*latency distance*). We now study how latency distance relates to distance in IP addresses (*address distance*), i.e., how clients far away in latency can be separated by splitting address space. We use the same definition of address distance used by Lee and Spring [29]. For each client in the pruned set, we choose the top 10% furthest clients in latency and refer to them as *distant clients*. Then, we calculate two address distances: 1) from the client in the pruned set to the distant clients and 2) between distant clients. For the first measure, we calculate the address distance between the client and each of the distant clients, and use the average as the address distance. For the second measure, we calculate the address distance between each pair of the distant clients and use the average as the address distance.

For existing aggregation methods, we find that about 37% to 47% of outliers are closer to the distant clients in address space than the distant clients are among themselves. Thus to separate them from the distant clients would require segregating them into their own, small aggregation units, e.g., /24s. For the remaining 53% to 63% of outliers, since they are further apart from the distant clients than the distant clients are apart amongst themselves, we could further divide the address space. In the case of over-aggregation, only 7% of clients are closer to the distant clients than the distant clients are apart amongst themselves, in address space. This means that for most clients in the case of over-aggregation, being far apart in latency to distant clients implies being far apart in address space. Thus, these clients should be easily separated from the distant clients by further dividing the address space. In the following, we introduce AP-atoms which takes advantage



(a) Single mode (b) Non-overlapping modes

(c) Overlapping modes (d) No mode

Figure 5: Four patterns of latencies

of the above observations and allow for flexible address space boundaries that adapt dynamically to network condition.

3. AP-ATOMS

In contrast to existing aggregation methods, AP-atoms are data-driven aggregation. It aggregates clients based on their latencies. Each AP-atom includes clients with similar latencies. The similarity in latencies, which determines the dispersion of clients, can be controlled depending on the requirements on the accuracy of client redirection. Since AP-atoms aggregate clients based on their latencies and the latencies of clients change with *network events*, e.g., route changes, AP-atoms dynamically adapt to changing network conditions, resulting in high-accuracy aggregation. Futher, AP-atoms are *server-independent*: the dispersion of aggregation units and of clients is small whichever server they are redirected to. To achieve server-independence, for each server, we find a set of aggregation units that have small dispersion to the server, which we call *AP-atom candidates*. Then, we merge the AP-atom candidates to obtain AP-atoms. In the following section, we first introduce how to obtain AP-atom candidates and then discuss how to merge AP-atom candidates.

3.1 AP-atom Candidates: Single-Server View

We start with the latencies between a server and its clients, which can be obtained, for example, from the TCP round-trip time estimates of the clients' connections at the server. Client latencies, observed over time, change with network events and thus have different patterns. We cluster clients based on patterns in their latency data and transform these client clusters into AP-atom candidates. In this section, we present patterns in latency data. We defer discussion on how to recognize latency patterns to the Appendix.

3.1.1 Latency Patterns

Before introducing patterns in latency data, we first discuss how to determine latency. If individual IP addresses have a large number of measured RTTs, the minimum or median RTT is commonly used as latency [15, 25]. However, to preserve client privacy, the finest address prefix granularity in our dataset is /24. As recent work has shown, it is possible for /24 address prefixes to include individual clients (with 32-bit IP addresses) that are dissimilar in latencies [21]. We use the modes of RTTs as latency. Since RTTs can have multiple modes, using modes instead of the minimum or median latency helps us distinguish dissimilar clients within the same /24 address block.

Figure 5 shows four patterns of RTT samples taken from /24 blocks. When individual clients in the same /24 block are similar in latencies, their combined RTTs either have a single mode as in Figure 5(a) or mutliple modes that do not overlap in time, as in Figure 5(b). A *single mode* occurs when RTTs are measured in a stable period of the network, where the network could be either uncongested or persistently congested. *Multiple non-overlapping modes* occur when RTTs are measured in a period including network condition changes, e.g., congestion or route changes. As individual clients are similar in latency, their RTTs change with the changing network conditions, resulting in non-overlapping multiple modes. Figure 5(c) shows a pattern including multiple modes that overlap in time, which we refer to as *overlapping modes*. Due to insufficient or noisy measurements, the modes of RTTs could be unidentifiable, which we refer to as *unidentifiable modes*. Figure 5(d) shows an example of noisy RTTs, where RTTs take a large range of values and thus no mode can be identified.

3.1.2 Clustering Clients Based on Latency Patterns

We want to cluster clients such that clients in the same cluster have similar latencies. The latency pattern of a client is identified for a given time period and can change over different time periods, depending on the network events in the periods. To ensure that clients that experienced the same network event, i.e., having the same latency pattern, can be found, we cluster clients based on latency patterns identified for the same time period. In the next section, we will discuss how to consolidate latency patterns over different time periods.

Once we have identified the latency patterns, we divide clients having the same latency pattern into the same group. For each group, we cluster clients based on the distance between their latencies. At any given time, the *distance between latencies* is the difference between the modes of latencies, the tolerance for which can be controlled by a pre-defined threshold. For clients with overlapping modes, since they already include individual clients with dissimilar modes, we consider that each

of them is a cluster by itself. Because /24 address prefixes are the finest granularity in our dataset, we cannot further split clients in such clusters even if they could have the distance between latencies larger than the predefined threshold. Nonetheless, since we can identify when a /24 client has overlapping modes, this problem can be further investigated by active probing individual clients in the /24 block.

For clients with non-overlapping modes, we determine if two clients should be in the same cluster by merging the RTTs of the two clients and then checking the latency pattern of the merged RTTs. If the merged RTTs continue to have non-overlapping modes, the two clients shoud be in the same cluster; otherwise, they are in different clusters. Two clients with non-overlapping modes are in the same cluster only if the modes of latencies of the two clients are similar both in values and in time duration. The degree of similarity is determined by the algorithm to identify latency patterns in the Appendix. To cluster clients with a single mode, we use the quality threshold (QT) clustering algorithm [28]. Since all clients only have a single mode, each of them is associated with the value of the mode and the QT algorithm clusters clients based on the values of their modes. The QT algorithm takes the pre-defined threshold as a parameter for clustering. After clustering, clients in the same cluster have difference in modes less than the predefined threhold.

Since the algorithms to identify latency patterns and the QT algorithm are of superlinear complexity [3, 11], clustering all clients together is of high complexity. We thus divide the Internet address space into prefixes of proper size, which we refer to as the *top prefix*, and cluster clients using the QT algorithm only within each top prefix, not across them. For the sizing of top prefixes, there is a tradeoff between scalability and computational complexity. As will be discussed in Section 3.1.4, the number of top prefixes affects the number of AP-atoms. Considering both scalability and computional complexity, we use /16s as top prefixes and need 45K /16s to cover the entire Internet address space in the 592K routable BGP prefixes provided by CAIDA.

3.1.3 Evolution of Client Clusters

Given a server, we can use latency patterns of clients in a single period to obtain the clusters of clients in that period (*single-period* clusters). Due to changing network conditions and the emergence of new clients, single-period clusters can only reflect the similarity of clients in one period. For each top prefix, we want a set of client clusters that dynamically adapts to the changing network conditions and cumulatively accommodates new clients. Further, since the latency patterns of clients could be misidentified in any one time period, we want to be able to correct the misidentification over time. We refer to such client clusters as *cumulative clusters*. As cumulative clusters use latencies of clients from both the past and current periods, cumulative clusters include a larger number of clients and more accurate latency patterns of clients.

Suppose we want to obtain the cumulative clusters in the $(n+1)$-st period. We first consolidate clients in the cumulative clusters in the n-th period together with clients in the single-period clusters in the $(n+1)$-st period, and then correct the latency patterns of clients that are misidentified. The cumulative clusters in the n-th period are obtained from the latency patterns of clients from the first to the n-th periods. When n is equal to 1, the cumulative clusters are the single-period clusters in the first period. Depending on whether a client has identifiable latency patterns on the n-th and $(n+1)$-st time periods, we have three cases (not including the case when the client has no identifiable latency patterns on both periods). For each case, we must treat clients differently based on their latency patterns.

In the first two cases, a client has an identifiable latency pattern in the $(n+1)$-st period. We cluster the client using the latency pattern and record the latency pattern for correction later. In the second case, a client has no identifiable latency pattern in the $(n+1)$-st period. We want to use the client's latency pattern in the n-th period to infer the one in the $(n+1)$-th period for the consolidation. More specifically, in the first two cases, if the client has overlapping and non-overlapping modes, the client is in the single-period cluster of the client in the $(n+1)$-st period. If the client has a single mode, we calculate the moving average of the mode and use the moving average to cluster the client with other single-mode clients later. Using moving average is to smooth the mode and thus be more resilient to atypical modes.

In the third case, if the client has overlapping modes in the n-th period, it is a cluster by itself in the $(n+1)$-st period. If the client has a single mode, the moving average of its mode in the n-th period carries to the $(n+1)$-st period and the moving average is used to cluster the client with other clients. If the client has non-overlapping modes, we check if the client is in the same cluster with other clients, referred to as *in-cluster clients*. If there is no in-cluster client, the client is a cluster by itself in the $(n+1)$-st period. If in-cluster clients exist, we use the most frequent pattern of these clients in the $(n+1)$-st period as the latency pattern of the client. It is possible that none of the in-cluster clients have identifiable pattern in the $(n+1)$-st period or the most frequent pattern still has non-overlapping modes. In this case, the client and its in-cluster clients are still in the same cluster in the $(n+1)$-st period. If the most frequent pattern has overlapping modes, the client is in a cluster by itself in the $(n+1)$-st period. If the

most frequent pattern has a single mode, the average mode of the in-cluster clients is used as the mode of the client in the $(n+1)$-st period. We calculate the moving average using the mode for the client and use the moving average to cluster the client with the other clients later.

After the operations above, clients in the two periods are consolidated. We then correct the latency patterns of clients to their most frequent latency patterns, but we do not correct the latency patterns of clients that have non-overlapping modes to the most frequent latency pattern, because non-overlapping modes are most likely temporary, due to network events. Since clients with non-overlapping and overlapping modes are already clustered, we must cluster clients with a single mode using the QT algorithm, which operates on the moving averages of their modes. All the resulting clusters combined comprise the cumulative clusters of clients in the $(n+1)$-st period. In the following, we use cumulative clusters to get AP-atom candidates and simply refer to cumulative clusters as clusters.

3.1.4 AP-atom Candidates: Optimal Prefix Splitting

After clustering, we obtain client clusters with similar latencies to a given server. However, these clusters only include clients whose latency patterns are identifiable in our dataset. There are still a significant portion of clients having unidentifiable latency patterns, which means we cannot directly use these clusters for global load balancing. Instead, we use them as a guide to generate a set of prefixes that cover the entire Internet address space.

Given a server, we can have a set of client clusters in each top prefix. Our goal is to find the minimum set of prefixes (i.e., AP-atom candidates) for each top prefix such that 1) these prefixes can cover the entire address space of the top prefix and 2) using the longest prefix match, clients in the same cluster are matched into the same prefix and clients in different clusters are matched to different prefixes. The longest prefix match algorithm [26], widely used for Internet routing, always finds the prefix that shares the most number of common bits with the address of the client. This property guarantees that even when one prefix is included in another prefix, the client can still be matched to the correct one. In the following, we refer to this problem as *optimal prefix splitting*.

To obtain the minimum set of prefixes, we construct a binary tree from all the prefixes covered by the address space of the top prefix. Each node in the tree is a prefix, where the root of the tree is the top prefix and the leaves are /24 clients. The children of a prefix $/x$ are two $/(x+1)$ prefixes that evenly split the $/x$ prefix. Clusters of /24 clients are indexed and each /24 client, if it has an identifiable latency pattern, is associated with the number of the cluster that the client is in. Clients in the same cluster have the same cluster number and must be matched to the same prefix. Clients with no identifable pattern has no cluster number and can be matched to any prefix depending on how the address space is split. Among all the nodes in the tree, we want to select the minimum number of nodes that can achieve the optimal prefix splitting, where the prefixes at the selected nodes comprise the minimum set of prefixes. For each node in the tree, we have two choices, either selecting the node or not.

Let us start the node selection from the top prefix. If the top prefix is not selected, the minimum set of prefixes is the union of the minimum sets of prefixes needed to achieve the optimal prefix splitting for the left and right children of the top prefix. If the top prefix is selected, there could be multiple clusters of clients in the subtree of the root node, where the *subtree of a node* consists of the node and all descendents of the node. We must determine which cluster of clients the top prefix should cover under the longest prefix match. Because clients in different clusters should be matched to different prefixes, each prefix can only cover one cluster. Suppose the top prefix is selected to cover the i-th cluster. To guarantee that clients in the i-th cluster are matched with the top prefix under the longest prefix match, for other prefixes we must select the largest ones whose subtrees do not include clients in the i-th cluster; otherwise, since these prefixes are smaller than the top prefix, clients in the i-th cluster in the subtree of these prefixes will be matched with them rather than the top prefix under the longest prefix match. We refer to such prefixes as *complementary prefixes* to the top prefix. To generalize the definition, the complementary prefixes to a prefix are the largest prefixes that are in the subtree of the prefix and do not include any client from the cluster covered by the prefix under the longest prefix match. When the top prefix is selected, the minimum set of prefixes to achieve the optimal prefix splitting is the top prefix and the union of the minimum set of prefixes needed for each complementary prefix to the top prefix, where the set of complementary prefixes depends on which cluster the top prefix covers.

From the selection process above, we can see that the problem of optimal prefix splitting can be divided into similar subproblems. We can use dynamic programming to solve the problem as follows. Suppose we have a node u and want to know the optimal splitting of the prefix at node u. We denote the minimum number of prefixes needed as $F_{opt}(u)$. The set of prefixes that achieve the minimum number of prefixes is the minimum set of prefixes. We denote the number of prefixes needed by selecting node u as $F_{in}(u,i)$, where i is the cluster number of the clients that the prefix at node u must cover under the longest prefix match. We index

clusters from 1 and use $F_{in}(u,0)$ for the case that no cluster is included in the subtree of node u. Suppose the i-th cluster is chosen to be covered by the prefix at node u. We denote the set of complementary prefixes to the prefix at node u as $P_{u,i}$. The subtree of each complementary prefix includes a set of clusters—the set may be empty. We denote the set of clusters in the subtree of node v as S_v, where S_v is empty if no cluster is in the subtree of node v. If node u is not selected, no cluster is covered by the prefix at node u and the optimal splitting is determined by the children of node u, denoted as C_u. We denote the number of prefixes needed by not selecting node u as $F_{out}(u)$. Thus, the optimal solution is:

$$F_{opt}(u) = \min\left\{\min_{i\in S_u}\left\{F_{in}(u,i)\right\}, F_{out}(u)\right\},$$

where $\min_{i\in S_u}\{F_{in}(u,i)\}$ is the minimum number of prefixes among all cases when different clusters in the subtree of node u are chosen to be covered. Based on the selection process, we also have that

$$F_{in}(u,i) = 1 + \sum\nolimits_{v\in P_{u,i}} F_{opt}(v)$$

and

$$F_{out}(u) = \sum\nolimits_{v\in C_u} F_{opt}(v).$$

We use dyanmic programming to compute the optimal solution. When traversing the binary tree, we stop at a node either when the subtree of the node includes no cluster or a single cluster. Suppose we stop at node v. If node v includes no cluster, we set $F_{in}(v,0) = 1$, because no further search is needed. If node v includes a single cluster and the cluster number is i, we set $F_{in}(v,i) = 1$. In both cases, we set $F_{out}(v)$ to ∞ such that the prefix at node v must be included to guarantee that either the entire address space is covered or each cluster is covered by a prefix. After we obtain the minimum set of prefixes, each prefix is an AP-atom candidate.

3.2 AP-atoms: Multi-Server View

We now have a set of AP-atom candidates for each server, where clients in the same AP-atom candidate have small distances between their latencies to the server. Since the sets of AP-atom candidates to different servers may be different, we next merge these sets to obtain AP-atoms. The goal of this merging operation is to achieve server-independence. Under the longest prefix match rule, the merging can be easily done by combining the set of AP-atom candidates of each server. Suppose we have a client that is in the prefix $a_1.b_1.c_1.d_1/x_1$ to server 1 and is in the prefix $a_2.b_2.c_2.d_2/x_2$ to server 2. For the two prefixes, since they cover the same client, the larger prefix must include the smaller one, i.e., the address space covered by the smaller prefix is within the address space covered by the larger one. After combining the sets of AP-atom candidates, we have both prefixes in the set of AP-atoms. Under the longest prefix match rule, the client is matched to the smaller prefix. Since the larger prefix includes the smaller one, the client is also in the larger prefix and thus has small dispersion to both servers.

Combining the set of AP-atom candidates to each server can give us the set of AP-atoms, but this set includes many *empty prefixes*, i.e., the ones that no clients are matched to. In other words, the address space covered by an empty prefix is the combination of the address spaces covered by other small prefixes. For instance, if the set of AP-atoms includes a $/x$ prefix and two $/(x+1)$ prefixes that evenly divide the address space of the $/x$ prefix, the $/x$ prefix is an empty prefix. After pruning all empty prefixes, we obtain the finalized set of AP-atoms.

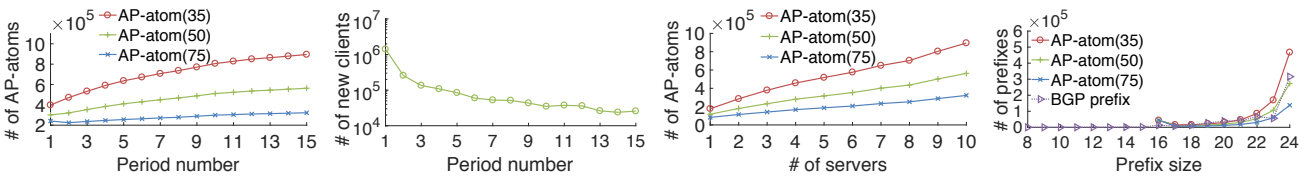
Figure 6: Locations of servers

4. PERFORMANCE EVALUATION

We use 30 days of latency measurements between clients and 10 servers, where the locations of servers are shown in Figure 6. Due to data sparsity, as discussed in the Appendix, we divide 30 days into 15 two-day periods and use measurements from two consecutive days to identify latency patterns. To present the capability of AP-atoms in trading off scalability for accuracy, we obtain three sets of AP-atoms using different pre-defined thresholds for the QT algorithm. The numbers of AP-atoms on these sets are chosen to be on scales that are larger than, equal to, and smaller than the number of aggregation units under existing aggregation methods respectively. Each set of AP-atoms is compared against existing aggregation methods in terms of scalability and server-independence.

4.1 Scalability of AP-atoms

We look at the scalability of AP-atoms in three aspects: 1) the scaling of AP-atoms over time, 2) the scaling of AP-atoms over the number of servers and 3) the distribution of prefix sizes of AP-atoms. The third aspect tells us the reasons for the advantages of AP-atoms over existing aggregation methods in terms of scalability.



(a) Scaling over time (b) New clients over time (c) Scaling over servers (d) Prefix size distribution

Figure 7: Scalability of AP-atoms

4.1.1 *Scaling of AP-atoms over Time*

Since AP-atoms evolve, we want to know the scaling of AP-atoms over time and the factors that affect the number of AP-atoms. Figure 7(a) shows the number of AP-atoms over 15 periods, where AP-atom(k) denotes the AP-atoms obtained by setting the pre-defined threshod equal to k ms, meaning that the radius of clusters in the QT algorithm is less than k ms. In general, as the period number increases, the total number of AP-atoms under all thresholds increases. The increment is mainly caused by the appearance of *new clients*, clients that never have an identifiable latency pattern in the previous periods. Depending on the latency patterns, new clients can cause the number of AP-atoms to increase in three ways: 1) If the new client has a single mode and the mode is far away from the modes of other clients, a separate cluster will be needed for the client. 2) If the new client has non-overlapping modes not similar to the latency patterns of other clients, the client will be in a separate cluster. 3) If the new client has overlapping modes, the new client will be in a separate cluster by itself.

Figure 7(b) shows the number of new clients in each period. In the first five periods, there are at least 100K new clients in each period and even until the last period, there are still about 20K new clients. When the number of new clients becomes stable at the 10-th period, the number of AP-atoms(50) and AP-atoms(75) increases slowly. In constrast, AP-atoms(35) are sensitive to new clients and increase with a rate of 15K per period after the 11-th period. Comparing AP-atoms(35) and AP-atoms(75), we see that a large threshold helps accommodate new clients without incurring a significant number of new AP-atoms. This is because most clients have a single mode, which can be easily accommodated with other clients when the threshold is large. When the last period ends, we obtain 895K AP-atoms(35), 563K AP-atoms(50) and 322K AP-atoms(75), which are 1.51×, 0.95×, and 0.54× the 592K number of BGP prefixes. To cover the address space of BGP prefixes, we need 693K /20 IP blocks. Since geo-blocks generally have smaller aggregation units than BGP prefixes, we need more geo-blocks to cover the entire address space.

It is noticeable that even when there are more than 100K new clients appearing in the second period, the number of AP-atoms(75) in the second period is still smaller than that in the first one. The decrement is caused by the correction of clients' latency patterns. From the first to the second period, clients identified to have non-overlapping and overlapping modes in the first period are corrected to have a single mode in the second one. Under a large threshold, all these clients that change to have a single mode can be clustered with other cilents, causing the total number of AP-atoms(75) to decrease. The latency patterns of such clients also change when the threholds are equal to 35 ms and 50 ms, but as AP-atoms(35) and AP-atoms(50) are more sensitive to new clients, the total still increases.

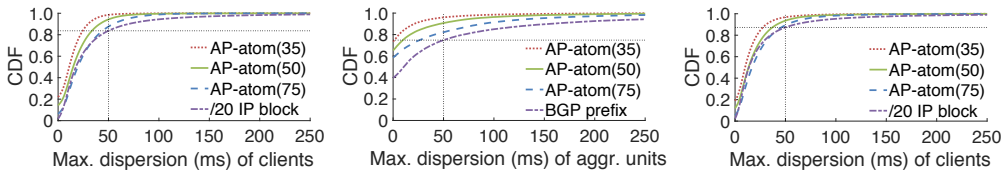
4.1.2 *Scaling of AP-atoms over the Number of Servers*

To be server-independent, AP-atoms are obtained by merging the AP-atom candidates of each server. We want to know the scaling of AP-atoms over the number of servers. Figure 7(c) shows the number of AP-atoms under different numbers of servers. As the number of servers increases, the number of clients accommodated into AP-atoms increases sharply (not shown here). At 8 servers, about 98% of clients have been accomondated into AP-atoms, but this does not slow down the linear increment of AP-atoms from server 9 to 10. The key reason for the increment is because AP-atom candidates of each server are obtained separately without considering others. More specifically, AP-atom candidates of a server are generated by the optimal prefix splitting based on the set of clients to the server, while servers have different sets of clients, which results in different sets of AP-atom candidates. An optimization across servers would further decrease the number of AP-atoms, which is a subject of our future research.

For small-scale CDNs with servers in 30 to 40 locations [2], the scaling of AP-atoms is not a concern. Even when the AP-atom candidates of all servers are used, based on the trend of AP-atoms(75) in Figure 7(c), the number of AP-atoms is approximately 470K for 40 servers, which is 0.8× the number of BGP prefixes. For large-scale CDNs with more than 1K locations of servers [14, 23], we can select servers whose views of AP-atom candidates differ significantly and only merge AP-atom candidates of these servers.

4.1.3 *Distribution of Prefix Sizes*

We have compared AP-atoms with existing aggrega-



(a) Accommodation of clients (b) Dispersion in the future (c) Dispersion in the future

Figure 8: Server-independence

tion methods in terms of the total number of aggregations needed to cover the Internet address space, but the total number cannot tell us the reasons for the scale of aggregation units. We look at the distribution of prefix sizes to understand the advantages of AP-atoms over existing aggregations. Figure 7(d) shows the distributions of prefix sizes of BGP prefixes and the three sets of AP-atoms, where the 1.6% of BGP prefixes smaller than /24 are not counted for the distribution. Since the IP2Location LITE database does not include the entire Internet address space, geo-blocks are not compared here. The largest prefix size of BGP prefixes is /8, while as AP-atoms are split from /16 top prefixes, AP-atoms have the largest prefix size equal to /16.

The three sets of AP-atoms include about 43K /16s, which are the 97% of all /16s we use to cover the entire address space. Large prefixes, due to covering large address space, are preferred to minimize the number of aggregation units. For each prefix size smaller than /16, AP-atoms(75) use a smaller number of prefixes than BGP prefixes and thus have a total number that is half the number of BGP prefixes. AP-atoms(50) have almost equal number of prefixes as BGP prefixes for each prefix size and thus have the same scale as BGP prefixes, while AP-atoms(75) include higher numbers of /23s and /24s than BGP prefixes. From the figure, we can see that the different scales of AP-atoms are mainly due to using different numbers of /23s and /24s. In other words, top prefixes generally must be split into small prefixes under a small threshold. It is interesting to note that only 5% of AP-atoms(50)'s /24 aggregation units prefixes are also BGP /24 prefixes. The /24 aggregation units in AP-atoms(50) are constructed based on path performance whereas /24 BGP prefixes reflect commercial organizational decisions that do not necessarily reflect different network performances.

4.2 Server-Independence

To be server-independent, an aggregation unit must have small dispersion to all servers, i.e., the maximum dispersion of the aggregation unit to all servers must be small. We have presented the maximum dispersion of existing aggregations in Section 2.1 and use the best existing aggregation to compare against AP-atoms. We first check if clients in the past and current periods have been properly accommodated in AP-atoms and then look at the performance of AP-atoms to deal with clients in future periods.

4.2.1 *Capability of Accommondating Clients*

AP-atoms accommodate clients based on the latency patterns of clients in the past periods. If clients are properly accommodated in AP-atoms, we would expect the maximum dispersion of the clients calculated using the latencies in the past periods to be small. We use the AP-atoms in the last period for the experiments. Since the maximum dispersion of /20 IP blocks is the smallest among existing aggregations (Figure 3(b)), we compare the three sets of AP-atoms with /20 IP blocks. Figure 8(a) shows the maximum dispersion of clients calculated using latencies from the first to the 15-th period. Compared to /20 IP blocks that have 17% of clients with maximum dispersion larger than 50 ms, AP-atoms(35) only have 1.5% of clients with the maximum dispersion larger than 50 ms. This implies that AP-atoms(35) properly accommodate clients in terms of latency. The main reason for AP-atoms(35) not able to control the dispersion of all clients less than 50 ms is because the threshold is 35 ms, which allows clients in the same AP-atom to have the maximum dispersion up to 70 ms. As the threshold increases to 75 ms, 12% of clients have maximum dispersion larger than 50 ms.

4.2.2 *Dispersion and Latency Prediction*

To use AP-atoms in practice, AP-atoms in the n-th period (including the n-th period) must be server-independent in the future $(n+1)$-st period. From Figure 7(b), we can see that the number of new clients starts to slow down from the 10-th period. We consider the 10-th period as the time that AP-atoms have accumulated enough clients. From the 10-th to the 15-th period, we use AP-atoms in the n-th period as the aggregation units in the $(n+1)$-st period and calculate the dispersion of clients in the $(n+1)$-st period. Given a server, the dispersion of a client to the server is the maximum dispersion of the client over all periods to the server. Using the dispersion of a client to each server, we obtain the maximum dispersion of the client to all servers.

Figure 8(b) shows the distribution of the maximum dispersion of aggregation units. Because the maximum dispersion of aggregation units in BGP prefixes is the smallest among existing aggregations (Figure 3(a)), we compare AP-atoms with BGP prefixes. While BGP

prefixes have 26% of aggregation units with dispersion larger than 50 ms, AP-atoms(35) can reduce such aggregation units to 4%. The aggregation units with zero dispersion are mainly /24s. It is interesting to note that the percentage of /24s in AP-atoms(75) is 12% less than that in BGP prefixes, but in Figure 8(b), the percentage of aggregation units with zero dispersion in AP-atoms(75) is higher than that in BGP prefixes. This is because /24s in AP-atoms are identified by their traffic in the past and clients with past history of traffic to a service provider are generally more likely to have traffic to the service provider in the future. Thus, /24s in AP-atoms(75) are more likely to have traffic than the /24s in BGP prefixes.

Figure 8(c) shows the distribution of the maximum dispersion of clients, where /20 IP blocks are compared with AP-atoms. About 13% of clients in /20 IP blocks have dispersion larger than 50 ms. Even with a scale of aggregation units that is 0.46× the number of /20 IP blocks, AP-atoms(75) can reduce this number to 10%. With a scale that is 1.3× the number of /20 IP blocks, AP-atoms(35) reduces this number to 3.5%.

5. RELATED WORK

In Internet mapping, clients are aggregated to aggregation units for scalable management of the Internet. Choosing the proper level of aggregation is a complex and crucial problem [6]. Exisiting aggregations generally aggregate clients based on several attributes: (1) geographic locations, (2) topological proximity, (3) LDNSs, or (4) fixed-size prefixes. In geoclustering, clients with geographic proximity are aggregated into geoclusters [27]. By considering BGP prefixes as a natural group of Internet clinets, geoclusters are obtained by splitting BGP prefixes into smaller prefixes until clients in prefixes have consensus on their geographic locations. Aggregating clients by topological proximity can be conducted at the prefix level or router level. Clients in the same BGP prefix largely have identical Internet routing. In anycast CDNs, clients are in the same BGP prefix are redirected to the same server and the nearest server is the one with the shortest BGP routing path to clients [4, 5, 22]. In iPlane, clients are aggregated into BGP prefixes to construct a topological map for the Internet [16, 17]. Moreover, in latency-based client redirection, the latency from the representitive client in a BGP prefix to a server is used as the latency from all clients in the prefix to the server [25]. The geographic locality of BGP prefixes has also been studied [24]. Recently, Lee and Spring [29] developed a method to aggregate /24s based on the last-hop routers or non-hierarchical relationships. However, since the method relies on topology discovery, which makes the aggregates hard to cover the entire Internet address space.

Aggregating clients by LDNSs is widely used in DNS-based CDNs [13,25], where clients using the same LDNS are redirected to the same server. As the DNS infrastructure evolves, the number of clients using remote DNS services grows, causing the mismatch problem between clients and LDNSs [10,20]. To solve this problem, end-user mapping directly uses the subnets of clients included in the EDNS queries to locate clients on the Internet [14]. However, end-user mapping requires the mapping system estimate the performance from servers directly to client aggregations [13]. Thus, a proper aggregation is crucial to both the scalability and accuracy of the mapping system. Using a heuristic method, Chen et al. [13] suggested that /20 IP blocks are a good tradeoff between scalability and accuracy.

Our work is also related to those studying whether /24s are a good aggregation unit. Krishnan *et al.* studied the similarity between individual clients in /24 prefixes in terms of geographic co-locality [21]. Lee and Spring studied the similarity between individual clients in /24 prefixes in terms of topological proximity [29]. Both works confirm that /24 prefixes are not necessarily the smallest aggregation units.

6. CONCLUSION

In this paper, we presented a comparative study of existing client aggregation methods and found that even for the best existing client aggregation, a significant portion (17%) of clients are far away from other clients in the same aggregation unit in terms of latency. We analyzed the causes for the widely dispersed clients and found the main cause to be that existing methods aggregate clients based on attributes other than path performance. To address this, we proposed a data-driven aggregation called AP-atoms, which group clients based on their path performance. The data-driven property enables AP-atoms to dynamically adapt to network changes. We studied the scalability of AP-atoms and compared AP-atoms against existing aggregation methods. Our results showed that AP-atoms can flexibly trade off scalability and accuracy. Using the same scale of aggregation as existing methods, AP-atoms can reduce widely dispersed clients by 2× and reduce the 98-th percentile difference in clients' latencies by almost 100 ms.

For future work, we want to further improve the scalability of AP-atoms by optimizing the set of AP-atom candidates across servers. Moreover, we want to use AP-atoms for performance prediction, which could significantly reduce active measurement loads and enable more intelligent mechanims to detect network congestions.

7. DATA DESCRIPTION AND PROCCESSING

Our dataset is provided by a commercial global load

Table 1: Coverage of RTT measurements

Country	/24s	AS	Provider
240/280	4.2M	17.6K/51.3K	160

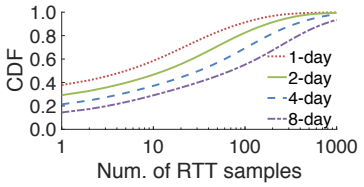


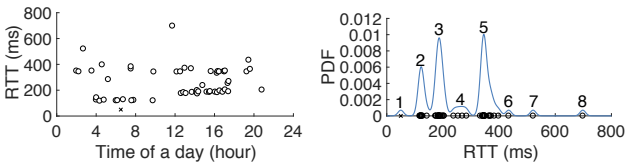
Figure 9: Distribution of the number of RTT samples in periods of different lengths

balancing service provider, which collects over 1 billion measurements per day, spanning more than 50,000 networks, covering over 100 participating providers, both public and proprietary, from a majority of countries in the world. Table 1 shows the coverage of RTT measurements in our dataset. Out of the 280 countries listed in ISO 3166, 240 are represented in the data. The data covers 34% of Autonomous Systems (ASs) in CAIDA's database [8], including all Tier-1 ASs, with 36% of the remainder being Tier-2 ASs. Clients in the data come from 4.2M /24s and have traffic with 160 providers. However, most providers do not have enough traffic with clients for analysis purposes and we only use the top 10 servers that have the most traffic with clients in our experiments.

Figure 9 shows the distribution of the number of RTT samples between all clients and servers in period lengths of 1 to 8 days. In 1-day periods, 38% of clients only have one sample. As the period length increases, the number of RTT samples from clients increases. Due to the *sparsity* of our dataset, we use 2-day periods for latency pattern identification in our experimentrs. We also include only clients with at least 5 measurements in a 2-day period.

7.1 Using the Modes of RTTs as Latency

Given a large number of RTT samples, the minimum RTT is generally used as latency to exclude queueing delay, while the median RTT is used to include queueing delay. We do not use the minimum RTT as latency because our dataset includes erroneously small measurements. Even after we prune RTTs less than the time required for signal to traverse the great-circle distance between clients and servers as in [25], the minimum RTT could still be much less than other measurements. Using the median RTT avoids erroneously small RTTs, but the queueing delay is a random component depending on network conditions. Further, both the minimum and median RTTs are a single value, which cannot identify /24 clients including individual clients with dissimilar latencies, while the modes of RTTs can



(a) RTT samples (b) MSC clusters

Figure 10: An example of applying KDE on RTT series to obtain MSC clusters.

reveal the patterns of latencies as shown in Figure 5.

We have obtained 2.3 millions of /24 clients that have an undentifiable mode in our dataset. When calculating dispersion, we only use clients that have a single mode and overlapping modes. We do not use clients with non-overlapping modes for simplicity of comparison, because when comparing non-overlapping modes from two clients, we must use modes from the same period of time, which are hard to estimate due to data sparsity. Since RTTs can only reflect the network conditions when the RTTs are measured, it is possible that RTTs are measured during temporary network congestions. To avoid using modes identified from these ayptical RTTs for dispersion calculation, we smooth the modes by their moving average and use the typical weights of moving average in TCP. We have verified that after moving average, among clients with large dispersion, using a different mode in other time periods only affects the disperison of less than 2% of the clients significantly.

7.2 Latency Pattern Indentification

In each time period, we want to identify the latency patterns of clients to a server by analyzing the measured RTTs between the clients and the server. To find latency patterns in measured RTTs, we use machine learning algorithms, i.e., mean shift clustering (MSC) [12] and total variation denosing (TVD) [19]. MSC is used to cluster RTTs, where RTTs in the same cluster are close in values and each cluster has a mode determined by the RTTs in the cluster. We refer to clusters obtained from MSC as the *MSC clusters*. We can use the relations between MSC clusters to identify latency patterns. TVD is used to determine the relations between MSC clusters, e.g., if the modes of MSC clusters are overlapping or non-overlapping in time. With TVD, we can automate the process of latency pattern identification.

7.2.1 *Identifying Latency Patterns with the Relations Between MSC Clusters*

We illlustrate how to identify latency patterns with MSC clusters using an example in Figure 10(a), which shows one-day RTT trace from a client havint overlapping modes to a server. At time 6, an erroneous RTT sample is marked as a cross. Before and after time 12,

there is a change of the minimum RTTs, which could be due to a network event. We use MSC to cluster RTTs based on their values and obtain the MSC clusters as in Figure 10(b). Each MSC cluster has a peak and the RTT corresponding to the peak is the *mode* of the cluster. MSC clusters 2, 3 and 5 have a much higher peak than other clusters, indicating the three clusters include a larger number of RTT samples. We refer to such MSC clusters as *significant clusters*. In contrast, the erroneous RTT, due to having a value much less than other RTTs, is isolated in MSC cluster 1. We refer to such MSC clusters as *insignificant clusters*. Other insignificant clusters include MSC clusters 4, 6, 7 and 8.

Let x and y be two MSC clusters. There are three relations between x and y: 1) x is *noise* to y, when x is an insignificant cluster and y is an significant cluster, e.g., MSC clusters 1 and 3; 2) x and y are *non-overlapping*, when x and y are significant clusters including RTTs that almost do not overlap in time, e.g., MSC clusters 2 and 3; 3) x and y are *overlapping*, when x and y are significant clusters including RTTs that overlaping in time, e.g., MSC clusters 2 and 5. Using the relations between MSC clusters, we can identify latency patterns. If RTT samples include significant clusters that are overlapping, the latency pattern has overlapping modes. If all significant clusters in RTT samples are non-overlapping, the latency pattern has non-overlapping modes. If there is only one significant cluster and other clusters are noise to the significant cluster, then the latency pattern has a single mode. The detailed procedure on latency pattern idenfication is as follows.

In the following, we use TVD to quantify the relations between MSC clusters.

Determining the Relations Between MSC Clusters Using TVD

The most well-known application of TVD is noise removal. Given a noisy signal as input, TVD recovers the original signal by smoothing out the noise. If the noisy signal is denoted as $\boldsymbol{r} = (r_1, \ldots, r_m)$, where r_i is the i-th element, the objective of TVD is to find a sequence $\boldsymbol{z} = (z_1, \ldots, z_m)$, a close approximation to the original signal by minimizing the following cost function,

$$\arg\min_{\boldsymbol{z}} \frac{1}{2}\sum_{i=1}^{m} |r_i - z_i|^2 + \lambda \sum_{i=2}^{m} |z_i - z_{i-1}|,$$

where λ is a tuning parameter penalizing the change to z_i. In this paper, we use TVD to identify the relations between MSC clusters as follows.

Given two MSC clusters x and y, we first set all RTTs in each MSC cluster to be the mode of that cluster and then merge the RTTs in the two clusters in the order of their measured time. Using the merged RTTs as input, $\boldsymbol{r}$, to TVD, we obtain the output $\boldsymbol{z}$ as previously described. Setting all RTTs in each cluster to the same value before applying TVD emphasizes the impact of one cluster on the other. If r_i is in x, $|z_i - r_i|$ is then the impact of RTTs in y on r_i after merging. Based on the impact of x and y on each other, we can determine the relation between x and y. Let us denote the change of r_i, i.e., $|z_i - r_i|$, as Δ_i and use a threshold Δ_{th} to determine if the change is significant. We must choose λ and Δ_{th} properly so that the relations between MSC clusters can be identified via TVD.

We can consider the merged RTTs in $\boldsymbol{r}$ as a sequence. In such sequences, RTTs in x and y are interleaved over time, which comprise various patterns and thus make it difficult to analytically determine λ and Δ_{th}. We view a sequence $\boldsymbol{r}$ as comprised of repeating *sub-sequences*, where each sub-sequence consists of three parts of RTTs with the second part seperating the first and third parts in time. RTTs in the first and third parts from one cluster and RTTs in the second part from the other cluster. Such sub-sequences can be combined together to constitute any form of sequences. Let us denote the part of RTTs in x and y in each sub-sequence are denoted as x_p and y_p and the second part as x_p, the first and third parts as y_p^l and y_p^r, which constitute y_p. For each sub-sequence, we study how to set λ and Δ_{th}, and how to use Δ_is to determine the relations between x_p and y_p. After that, we extend the relations between x_p and y_p in sub-sequences to the relation between x and y.

In each sub-sequence, the relation between x_p and y_p determines the values of Δ_i's. We thus want to use the values of Δ_i's to determine the relation between x_p and y_p. Let the number of RTTs in cluster w be $N(w)$. When x_p is noise to y_p, we expect the number of RTTs in x_p to have no statistical significance. In other words, x_p includes RTTs that occur with low probability, such as RTTs due to transient congestions or measurement errors. We use a threshold N_{lb} to determine if an MSC cluster includes statistically significant RTTs and consider a cluster with less than N_{lb} as insignificant. Let d_x and d_y be the modes of x and y, and $\mathbb{1}(x)$ be an indicator function equal to 1 if $x > 0$ and -1 if $x < 0$. We have the following theorem for the optimal solution to TVD.

THEOREM 1. *Let the input $\boldsymbol{r}$ consists of RTTs in x_p and y_p. For any $\lambda \in (0, |d_x - d_y|/2)$, we have 1) if both y_p^l and y_p^r are not empty, the solution that minimizes the cost function is $z_i = r_i + \Delta_i \mathbb{1}(d_x + d_y - 2r_i)$, where $\Delta_i = \frac{2\lambda}{N(x_p)}$ for all r_i's in x_p, $\Delta_i = \frac{\lambda}{N(y_p^l)}$ for all r_is in y_p^l, and $\Delta_i = \frac{\lambda}{N(y_p^r)}$ for all r_i's in y_p^r; 2) if either y_p^l or y_p^r is empty, the optimal solution is reached when $\Delta_i = \frac{2\lambda}{N(x_p)}$ for all r_i's in x_p and $\Delta_i = \frac{2\lambda}{N(y_p)}$ for all r_i's in y_p.*

PROOF. The proof is provided in Appendix .1. □

From Threorem 1, we can see that the number of RTTs in x_p determines the value of Δ_i's of r_i's in x_p. Given N_{lb}, by setting λ to $N_{lb}\Delta_{th}$, we can have that when x_p is insignificant and has less than N_{lb} RTTs, e.g., x_p is noise to y_p, Δ_i is larger than Δ_{th} for all r_i's in x_p and when x_p is significant and has more than N_{lb} RTTs, e.g., x_p is overlapping to y_p, Δ_i is smaller than Δ_{th} for all r_i's in x_p. This implies the relation between x_p and y_p reflects on the relative values between Δ_i's and Δ_{th}. We thus want to define the relation between x_p and y_p as follows: 1) When $\Delta_i > \Delta_{th}$ for all r_i's in x_p and $\Delta_i < \Delta_{th}$ for all r_i's in y_p, we consider all RTTs in x_p have a significant change, while no RTTs in y_p have a singnificant change, corresponding to that x_p is noise to y_p. This condition holds only if both y_p^l and y_p^r are not empty; 2) When $\Delta_i < \Delta_{th}$ for all r_i's in x_p and $\Delta_i < \Delta_{th}$ for all r_i's in y_p, we consider all r_i's in x_p and y_p to have no significant change, corresponding to that x_p and y_p are non-overlapping. Meanwhile, since r_i's in x_p and y_p have $\Delta_i < \Delta_{th}$, we have $N(x_p) > N_{lb}$ and $N(y_p) > N_{lb}$, which means both x_p and y_p are significant; 3) For other cases of RTT changes, we consider x_p and y_p to be overlapping.

After quantifying the relation between x_p and y_p in sub-sequences, we discuss how to extend such relations to the relation between x and y. A merged sequence of x and y may include sub-sequences where x_p and y_p have different relations. To handle the different relations of sub-sequences, we introduce two parameters P_x and P_y, the percentage of RTTs in a sequence that experience significant changes larger than Δ_{th} in x and y respectively after merging. If a sequence only consists of sub-sequences where x_p is noise to y_p, we have $P_x = 1$ and $P_y = 0$. For sequences consisting of sub-sequences of different relations, we have that P_x and P_y are between 0 and 1.

We define two thresholds h_1 and h_2 to determine if P_x and P_y are close to 0 or 1, where $h_1 < h_2$, e.g., 0.2 and 0.8. Then, we can extend the relations between x_p and y_p to the relation between x and y as follows: 1) If $P_x < h_1$ and $P_y > h_2$, we consider most r_i's in x do not have a significant change and most r_i's in y have a significant change, i.e., y is noise to x. Similarly, if $P_y < h_1$ and $P_x > h_2$, x is noise to y; 2) If $P_x \leq h_1$ and $P_y \leq h_2$, both RTTs in x and y do not have a significant change, i.e., x and y are non-overlapping; 3) Otherwise, x and y are overlapping. We will discuss how to choose h_1 and h_2 in Appendix .2.

APPENDIX

.1 Proof of Theorem 1

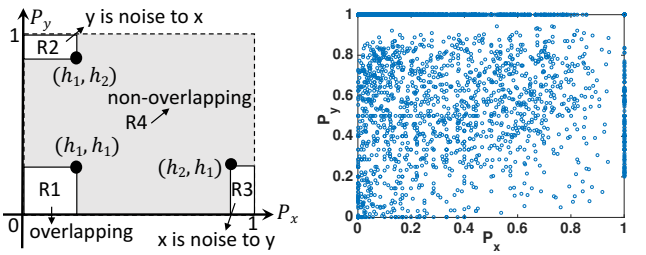
Without loss of generality, we asume $d_x < d_y$. When the optimal solution is reached, for any $i \in \{1, \ldots, m\}$, we must have $d_x \leq z_i \leq d_y$. For z_i's less than d_x or larger than d_y, we can always get a smaller cost by setting these z_i's to d_x or d_y whichever is closer. Similarly, when the optimal solution is reached, for z_i's in the same part (y_p^l, x_p or y_p^r), we must not have oscilating Δ_i's, e.g., $\Delta_2 > \Delta_1$ and $\Delta_2 > \Delta_3$. For such Δ_2's, a smaller cost can be achieved by setting $\Delta_2 = (\Delta_1 + \Delta_3)/2$. Thus, for all z_i's in the same part, we have a monotonic sequence of Δ_i's. Assuming the sequence of Δ_i's in the same part is non-decreasing, we can have $|z_i - z_{i-1}| = |\Delta_i - \Delta_{i-1}| = \Delta_i - \Delta_{i-1}$ for r_i's in each part and rewrite the cost function as

$$f = \sum_{i=1}^{m} \Delta_i^2 + \lambda \left(d_y - d_x - \Delta_1 + \sum_{i=N(y_p^l)+2}^{m} |z_i - z_{i-1}| \right).$$

When f is minimized, we have that for each r_i where $i \in \{2, \ldots, N(y_p^l)\}$, $\partial f / \partial \Delta_i = 2\Delta_i = 0$. However, since the sequence of Δ_i is non-decreasing, we have that f is minimized when $\Delta_i = \Delta_1$ for $i \in \{2, \ldots, N(y_p^l)\}$. This implies that the cost is minimized when all r_i's in the same part experience the same change. By setting Δ_i's in y_p^l, x_p and y_p^r to $\Delta_{y_p^l}$, Δ_{x_p} and $\Delta_{y_p^r}$ respectively, we have

$$f = \frac{1}{2}N(y_p^l)\Delta_{x_p^l}^2 + \frac{1}{2}N(x_p)\Delta_{x_p}^2 + \frac{1}{2}N(y_p^r)\Delta_{y_p^r}^2 + \lambda \left(2(d_y - d_x) - \Delta_{y_p^l} - \Delta_{y_p^r} - 2\Delta_{x_p} \right).$$

Taking the derivatives of f with respect to $\Delta_{x_p^l}$, Δ_{x_p} and $\Delta_{y_p^r}$, we have that when the optimal solution is achieved when $\Delta_{y_p^l} = \frac{\lambda}{N(y_p^l)}$, $\Delta_{y_p^r} = \frac{\lambda}{N(y_p^r)}$, and $\Delta_{x_p} = \frac{2\lambda}{N(x_p)}$. Following the same procedure, we can prove the optimal solution when either y_p^l or y_p^r is empty.

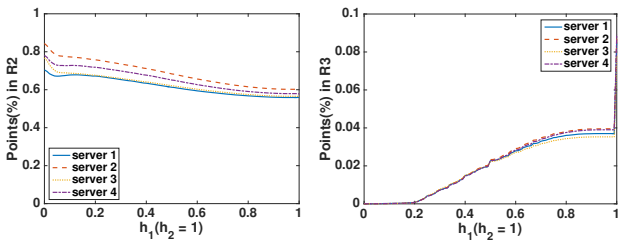


(a) Four relations between x (b) Distribution of p_x and p_y and y

Figure 11: Illustration of the four relations between MSC clusters x and y, and an example of the distribution of P_x and P_y.

.2 Tuning Parameters h_1 and h_2

In Figure 11(a), the four relations between x and y are represented as four regions in a plane with P_x- and P_y-axes. When point (P_x, P_y) is in R1, we have $P_x < h_1$ and $P_y < h_2$, which means x and y are overlapping. We want to set h_1 and h_2 to correctly identify the relation between x and y.



(a) Percentage of points in R2

(b) Percentage of points in R3

Figure 12: The choice of h_1 and h_2

The values of h_1 and h_2 depend on the patterns of measurements in our dataset. We randomly select 4 servers and study the RTT series between these servers and rAPs. For each RTT series, following the procedure of AP type identification, we calculate the values of P_x and P_y. The distribution of P_x and P_y with respect to a given server is shown in Figure 11(b), from which we can see that there is a large number of points at $P_y = 1$ near R2, but no points where $P_x = 1$ near R3. This is because in the AP identification procedure above, we always first select an n-cluster and check the relation between the first n-cluster and other MSC clusters, i.e., we use the first n-cluster as x and other clusters as y. Compared to n-clusters, other MSC clusters largely include much less RTT measurements. We can thus expect that the number of points in R2 to be large and in contrast, the number of points in R3 to be small. Next, we quantitatively study the number of points in R2 and R3.

By setting h_2 equal to 1, i.e., R2 becomes a line with length h_1, we study the percentage of points in R2 over all points with P_x less than h_1 for the 4 servers under various values of h_1. In Figure 12(a), it can be seen that for all servers, about 60% of points are included in R2 regardless of h_1. Considering that a majority of points can be coverd by R2 when h_2 is equal to 1, the ideal value for P_y when y is noise to x, we set h_2 to 1. Since h_2 is set to 1, R3 becomes a line with length h_1. In Figure 11(b), we can clearly see that there is a boundary between points near R3 and other regions on line $P_x = 1$. Similarly, under various values of h_1, we show the percentage of points in R3 over all the points with P_y less than h_1 for all servers. In Figure 12(b), we can see that the boundary is $h_1 = 0.2$. We thus set h_1 to 0.2 so that R3 contains a relatively small number of points.

A. REFERENCES

[1] Extended version. `https://www.dropbox.com/s/s0rltgfupl6n4uk/ap-atom-extended.pdf?dl=0`.

[2] USC CDN coverage. `http://usc-nsl.github.io/cdn-coverage`.

[3] Anthony Danalis, Collin McCurdy, and Jeffrey S. Vetter. Efficient quality threshold clustering for parallel architectures. In *Parallel and Distributed Processing Symposium*, pages 1068–1079, 2012.

[4] Ashley Flavel, Pardeepkumar Mani, David A. Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern CDNs. In *Proc. NSDI*, pages 381–394, 2015.

[5] Ashley Flavel, Pradeepkumar Mani, and James Miros. System and Method for Content Delivery Using Dynamic Region Assignment. US Patent No. 20120290693, 2011.

[6] Bruce M. Maggs and Ramesh K. Sitaraman. Algorithmic nuggets in content delivery. In *ACM SIGCOMM CCR*, pages 52–66, 2015.

[7] CAIDA. The CAIDA AS Relationships Dataset, <May 2015>. `http://www.caida.org/data/as-relationships/`.

[8] CAIDA. Routeviews Prefix to AS mappings Dataset for IPv4 and IPv6. `http://www.caida.org/data/routing/routeviews-prefix2as.xml`.

[9] Carlo Contavalli, Wilmer van der Gaast, David Lawrence, and Warren Kumari. Client Subnet in DNS Queries. RFC 7871, May 2016.

[10] Cheng Huang, Ivan Batanov, and Jin Li. A practical solution to the client-LDNS mismatch problem. In *ACM SIGCOMM CCR*, 2012.

[11] Daniel Freedman and Pavel Kisilev. Fast mean shift by compact density representation. In *Computer Vision and Pattern Recognition*, pages 1818–1825, 2009.

[12] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[13] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The Akamai Network: A Platform for High-Performance Internet Applications. In *Proc. SIGOPS*, pages 2–19, 2010.

[14] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. In *Proc. SIGCOMM*, pages 167–181, 2015.

[15] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *Proc. SIGCOMM*, pages 15–26, 2004.

[16] Harsha V. Madhyastha, Ethan Katz-Bassett, Thomas Anderson, Arvind Krishnamurthy, and

Arun Venkataramani. *iPlane Nano*: Path prediction for peer-to-peer application. In *Proc. NSDI*, pages 137–152, 2009.

[17] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, and Colin Dixon. *iPlane*: An information plane for distributed services. In *proc. OSDI*, pages 367–380, 2006.

[18] IP2Location. Geolocate IP Address Location using IP2Location. `http://www.ip2location.com`.

[19] Ivan Selesnick. Total variation denoising (an MM algorithm). `https://goo.gl/rjBpG`, 2014.

[20] John S. Otto, Mario A. Sánchez, John P. Rula, and Fabián E. Bustamante. Content delivery and the natural evolution of DNS. In *Proc. IMC*, 2012.

[21] Manaf Gharaibeh, Han Zhang, Chritos Papaopoulos, and John Heidemann. Assessing co-locality of IP blocks. In *IEEE Global Internet Symposium*, 2016.

[22] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of anycast CDN. In *Proc. IMC*, pages 531–537, 2015.

[23] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. Mapping the expansion of Google's serving infrastructure. In *Proc. IMC*, pages 313–326, 2013.

[24] Michael J. Freedman, Mythili Vutukuru, Nick Feamster, and Hari Balakrishnan. Geographic locality of IP prefixes. In *Proc. USENIX ATC*, pages 153–158, 2005.

[25] Rupa Krishnan, Harsha V. Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize CDN performance. In *Proc. IMC*, pages 190–201, 2009.

[26] Tsunemasa Hayashi and Toshiaki Miyazaki. High-speed table lookup engine for IPv6 longest prefix match. In *Proc. IEEE GLOBECOM*, pages 1576–1581, 1999.

[27] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for Internet hosts. In *Proc. SIGCOMM*, pages 173–185, 2001.

[28] Xin Jin and Jiawei Han. Quality threshold clustering. *Encyclopedia of Machine Learning, Springer*, pages 820–820, 2011.

[29] Youndo Lee and Neil Spring. Identifying and aggregating homogeneous IPv4 /24 blocks with Hobbit. In *Proc. IMC*, pages 151–165, 2016.