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

Yibo Pi, Sugih Jamin, Peter Danzig, and Jacob Shaha

***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 problem, which determines the number of aggregation units that a mapping system has to maintain and 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 trade off scalability for accuracy and adapt for changing network conditions. Since AP-atoms are obtained from the passive measurements of existing traffic between server providers and clients, no extra measurement overheads are incurred. Our experiments show that by using the same scale of client aggregations, AP-atoms can reduce the number of widely dispersed clients by almost 2× and the 98th percentile difference in clients' latencies by almost 100 ms.**

***Index Terms*— Client aggregation, global load balancing, data-driven, scalability and accuracy.**

I. Introduction

DNS-BASED client redirection has been adopted by many CDN provides like Google [26] and Akamai [11]. 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 needs 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 remote to the clients [12]. 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 in the Internet using the /24 subnets of the clients rather than their DNS resolvers. End-user mapping is made possible by the recent extension to the DNS protocol (EDNS), allowing recursive nameservers to carry the subnet of clients in their DNS queries [6]. 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 [12]. Clients can also be aggregated by their geographic locations [29] 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. By studying the causes for widely dispersed clients, we find that the wide dispersal of client latencies in existing aggregations are caused by aggregating clients based on attributes other than path performance. To address this, we propose *AP-atoms*. AP-atoms are data-driven and group clients based directly on their path performance (e.g., latency) to servers.[1] Since the path performance between clients and servers can be passively measured from user traffic available to service providers [14], AP-atoms can be obtained without incurring extra measurement overheads. To obtain AP-atoms, we use machine learning algorithms to identify distinct latency patterns and cluster clients based on these patterns. Besides the next-generation mapping system, AP-atoms also have potential uses in other emerging applications, e.g., consumer cloud storage [15], that could benefit from the scalable and accurate estimation of path performance between clients and servers.

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 trade off scalablity for accuracy. The method relies on the passive measurements of existing traffic between service providers and clients and thus incurs no extra measurement overheads. The data-driven property

Manuscript received February 25, 2018; revised May 19, 2018 and August 21, 2018; accepted October 4, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor P. P. C. Lee. Date of publication November 9, 2018; date of current version December 14, 2018. *(Corresponding author: Yibo Pi.)*

Y. Pi and S. Jamin are with the Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: yibo@umich.edu; sugih@umich.edu).

P. Danzig is with Panier Analytics, Menlo Park, CA 94025 USA.

J. Shaha is with the United States Military Academy, West Point, NY 10996 USA.

Digital Object Identifier 10.1109/TNET.2018.2878019

[1] In the literature, performance measurements are commonly used to estimate performance between servers and pre-determined client aggregations for better load balancing [9], [26], but not used to obtain client aggregations.

enables our mechanism to dynamically adapt to changing network conditions.

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

II. 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 [12], [20], [24], [29], 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 sites of CDN and cloud service providers, including Akamai, Amazon, Google, Level3 and Microsoft Azure, and others. Our dataset only includes measurements on network performance (e.g., RTT and throughput) and does not contain any sensitive information about clients (e.g., traffic content and passwords). Further, individual addresses (32-bit addresses) are anonymized with the last byte of addresses masked for privacy. In other words, addresses in our dataset are represented as /24 IP blocks. More details on our dataset are presented in Appendix A.

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* [29]. 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 [12]. 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 [12], [26].

Using our dataset, we simulate the use of aggregation methods above in real systems. In Internet routing, routers choose next hops 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. 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. We study the performance of existing client aggregations in terms of latency. The same concept of dispersion applies to other performance metrics such as bandwidth and packet loss.

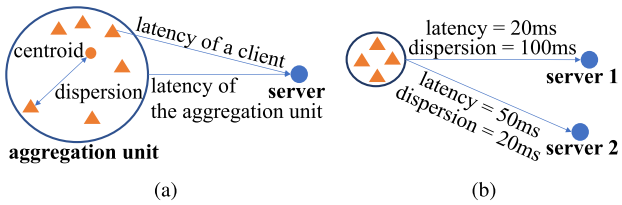

Fig. 1. Illustration of latency dispersion. (a) Latency dispersion of clients. (b) Impact of dispersion.

A. 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 [12], [26]. To understand the effects of using latency of aggregation units to perform redirection for all clients inside the unit, we define two metrics for both clients and aggregation units 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. The dispersion of clients tells us how many clients are far away from other clients in the same aggregation unit in terms of latency. The *dispersion of an aggregation unit to a server* is the largest dispersion among all clients in the aggregation unit, which tells us the worst performance of existing aggregation units. We use the largest dispersion instead of a percentile-based one because the sizes of aggregation units range from a few to thousands of clients and the percentile-based dispersion is biased against small ones, where each client takes a significant portion of its aggregation unit.[3] More importantly, using largest dispersion gives us a worst-case performance for aggregation units. We use largest dispersion only in comparing the performance of aggregation methods, not part of aggregation methodology.

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. We want to use latency dispersion as a metric to indicate the quality of aggregation units. Small latency dispersion enables an accurate estimation of latency to aggregation units and thus an accurate server redirection. Since clients in an aggregation unit could be redirected to mutliple servers, we prefer that the dispersion of the aggregation unit to all possible servers be small. Although latency dispersion may

[2]In [26], one client in the aggregation unit is selected randomly and its latency is considered as representative of the unit.

[3]For large aggregation units, we could have a small portion of clients far away from others, but we do not consider them as outliers, in contrast, representatives of a small group of clients that are aggregated improperly. The reasons for this are that 1) the clients we can use to calculate dispersion for large aggregation units only takes a small portion of all clients in the units, which implies that the actual dispersion could be larger, and that 2) the latencies of clients are accurately identified as shown in Section IV-C.

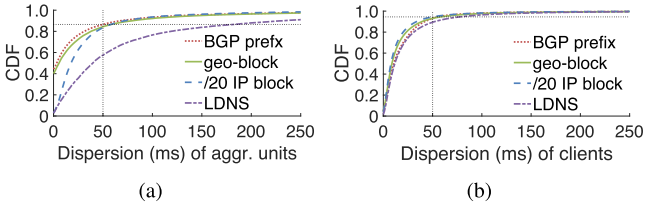

Fig. 2. Latency dispersion to a single server.

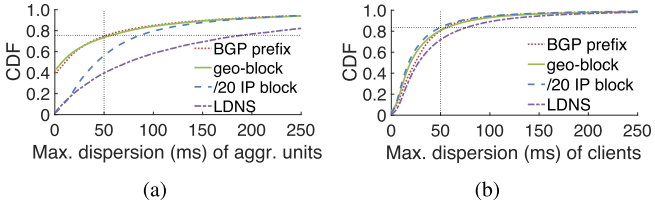

Fig. 3. Maximum latency dispersion to multiple servers.

give us false positive results[4] in some cases, these cases cover at most 0.8% of clients for all aggregation methods. In the following, we first show the latency dispersion of existing client aggregations to a single server and the maximum latency dispersion to all servers.

1) Latency Dispersion to a Single Server: We use the database of routable BGP prefixes from CAIDA [4] for aggregation by BGP routing paths and the IP-to-location database from IP2Location [17] for aggregation by geographic locations. We use $/20$ IP blocks for fix-sized prefixes and aggregate clients by their LDNSs. Addresses in the IP-to-location database are represented as IP blocks and each IP block has an estimated location. For comparison with other aggregations, we aggregate IP blocks into geo-blocks that have a distribution of prefix size similar to BGP prefixes, where IP blocks in the same geo-block have locations in a circle with a radius less than 200 miles. We compare the latency dispersion of the four aggregation methods above on two granularities: aggregation units and clients. Since the last byte of addresses in our dataset is anonymized for privacy reasons, *each client in our dataset is a /24 IP block*, which is actually an aggregation of individual addresses (32-bit).

To calculate latency dispersion, we use the latency measurements from all clients to one server.[5] Among all clients, we have obtained 1.7 million clients that have stable latencies (i.e., the single-mode latency in Figure 5(a)) to the server. For comparison among aggregation methods, we group these clients to aggregation units for each aggregation method respectively. For LDNS, clients using the same LDNS are in the same aggregation unit. For BGP prefixes, /20 IP blocks and geo-blocks that are represented as prefixes, we use the longest prefix match to group clients. Each client is mapped to the aggregation unit having the longest common prefix with the client's address. Then, we calculate the dispersion of clients and aggregation units. Figure 2(a) shows the cumulative distribution function (CDF) of latency dispersion of *aggregation units* to the 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 [26], 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. In Figure 2(a), BGP prefixes and geo-blocks include a portion of aggregation units with zero dispersion. This is due to half of aggregation units from BGP prefixes and geo-blocks being of size /24, comprising of only one client each.

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 [3], 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 clients to all 10 servers. Among all clients, we have obtained 2.3 million clients having stable latencies to at least one server. 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,[6] 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%

[4]False positive results occur when clients in an aggregation unit use the same gateway (not middleboxes, see Section III-C1) to the rest of the Internet and have different path performance. Such clients would have large dispersion (larger than 50 ms) to all servers.

[5]Using latency measurements to other servers gives us similar results, not presented here. We do not average latency dispersion of clients over servers because each client has communications with a different set of servers.

[6]Since geo-blocks and BGP prefixes are similar in the distribution of prefix sizes, they are close in dispersion in Figure 2 and 3.

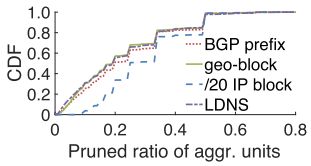

Fig. 4. Distribution of the pruned ratio.

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.

B. Causes for Large Dispersion

An aggregation unit has large dispersion when a small portion of clients (*minorities*) have latencies significantly different from other clients in the same aggregation unit,[7] or when the aggregation unit is overlarge and includes clients that should be divided into smaller aggreagation units (*over-aggregation*).

1) Identifying Minorities 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 total number of clients (*pruned ratio*).

Figure 4 shows the distribution of pruned ratios for the four aggregation methods. We consider an aggregation unit as containing *minorities* 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 aggregation methods, only 25% to 31% of their aggregation units have a pruned ratio less than 0.1. Since small aggregation units include a fewer number of clients, they are less likely to have a small pruned ratio. Among aggregation units with a pruned ratio less than 0.1, over 84% of them are of size at least /18 for BGP prefixes and geo-blocks. If we now consider aggregation units with a pruned ratio larger than 0.2 to be due to *over-aggregation*, over 40% of aggregation units have large dispersion due to over-aggregation for all aggregatiom methods. Since each client takes a significant portion in small aggregation units, small aggregation units with large dispersion are more likely 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.

2) Inflexibility of Existing Aggregations: Existing aggregation methods partition Internet address space by clustering clients similar in certain attributes. The attributes do not necessarily reflect path performance of clients. Moreover, given an attribute by which clients are aggregated, partitioning of the address space is fixed, not adaptive to changing network conditions. This inflexibility of partitioning causes the minorities and over-aggregation problems. In this section, we analyze the relationship between inflexible address space partitioning due to the use of arbitrary, non-performance related attributes and the presence of minorities and over-aggregation.

Minorities and over-aggregation are due to clients' having large distances in their latencies (*latency distance*). We now study how latency distance relates to the 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 [31]. 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 minorities 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., /23s. For the remaining 53% to 63% of minorities, 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 of the above observations and allow for flexible address space boundaries that adapt dynamically to network condition.

III. 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, we find a set of aggregation units that have small dispersion from the view of each single

[7]We have verified that atypical latencies are not the reason to large dispersion. Among aggregation units with large dispersion, less than 2% of them include minorities that could experience *atypical* latencies (20 ms larger than latencies in neighboring periods) such as caused by network congestion.

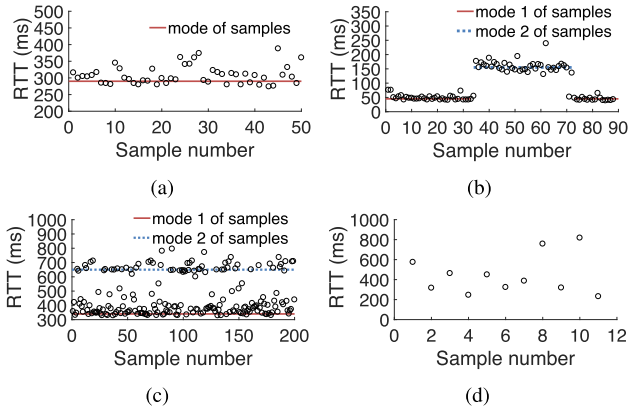

Fig. 5. Four patterns of latencies. (a) Single mode. (b) Non-overlapping modes. (c) Overlapping modes. (d) No mode.

server, referred to as *AP-atom candidates*. Then, we merge the view of each server (i.e., AP-atom candidates) to obtain AP-atoms.

A. AP-Atom Candidates: Single-Server View

AP-atom candidates are generated from clusters of clients with similar patterns in their latency measurements. Before discussing how to cluster clients based on their latency patterns, we first present latency patterns and the idenfication of latency patterns. 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.

1) Latency Patterns: Before introducing patterns in latency measurements, we first discuss how to determine latency. When an individual address (a 32-bit IP address) has a large number of RTTs available [13], [26], the minimum or median RTT is generally used as latency. However, when this method is used for passive measurements, it faces two problems: 1) a significant portion of individual addresses could have an insufficient number of measurements in passive measurements,[8] and 2) the measurements of individual addresses could be inflated due to network congestions, not representative for typical latencies. To solve these problems, we aggregate RTTs and use the collective behaviors in the aggregated RTTs to determine latency. As the finest address prefix granularity in our dataset is /24 for privacy purposes, we aggregate individual addresses in the same /24 IP block. As recent work has shown, it is possible for /24 IP blocks to include individual addresses that are dissimilar in latencies [21]. We use the modes of RTTs as latency, where the modes of RTTs are the prevalent RTTs (see Section III-A2 for more details). Since RTTs can have multiple modes, using modes instead of the minimum or median latency helps us distinguish dissimilar addresses within the same /24 IP block.

Figure 5 shows four patterns of RTTs taken from a client (i.e., a /24 IP block). When individual addresses 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 addresses 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, referred 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.

[8]An individual address could easily have sparse data in passive measurements due to 1) the individual address may have infrequent or no communications with the servers of interest and 2) latency can be measured passively only at limited stages of communications, e.g., during TCP's three-way handshake [14].

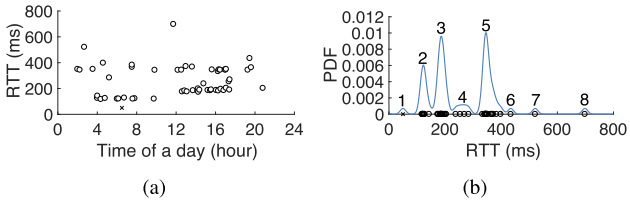

Fig. 6. An example of applying MSC to obtain MSC clusters. (a) RTT samples. (b) MSC clusters.

2) Identifying Latency Patterns: We identify latency patterns using two machine learning algorithms, i.e., mean shift clustering (MSC) [10] and total variation denosing (TVD) [18]. MSC is used to cluster RTT samples, where samples in the same cluster (called *MSC cluster*) are close in values. TVD is used to determine the relations between MSC clusters. In Figure 6(a), we use an example to show how to obtain MSC clusters and the relations between MSC clusters. The example shows one-day RTT trace from a /24 block to a given server. At time 6, an erroneous sample, much less than other samples, 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. After applying MSC on the RTT samples, we obtain the MSC clusters in Figure 6(b). The erroneous sample is isolated in MSC cluster 1. 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 that they include a much larger number of samples.

To distinguish between MSC clusters (denoted as x and y), we identify three relations: 1) x is *noise* to y, i.e., x has a much less number of RTTs than y, and RTTs in x and y are interleaved in time, e.g., MSC clusters 1 and 5; 2) x and y are *non-overlapping*, i.e., x and y are not noise to any other clusters and their RTTs almost do not overlap in time, e.g., MSC clusters 2 and 3; 3) x and y are *overlapping*, i.e., x and y are not noise to any other clusters and their RTTs are interleaved in time, e.g., MSC clusters 2 and 5. Only the modes of MSC clusters that are not noise to any other clusters are considered stable and used to calculate latency dispersion in experiments. In the following, we use TVD to quantitatively determine the relations between MSC clusters.

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 $\boldsymbol{z} = (z_1, \ldots, z_m)$, an approximation to the original signal by minimizing the following cost function,

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

where λ is a tuning parameter penalizing the change to z_i. 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 both 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 TVD. 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.

After TVD, we calculate the percentages of RTTs having significant changes in x and y, denoted as P_x and P_y respectively. Let h be a threshold to determine if most RTTs have significant changes. The relations between x and y are summarized as follows: 1) If $P_x \leq 1 - h$ and $P_y \geq h$, most RTTs in y have significant changes, while most RTTs in x do not have significant changes, i.e., y is noise to x. Similarly, if $P_y \leq 1 - h$ and $P_x \geq h$, x is noise to y; 2) If $P_x \leq 1 - h$ and $P_y \leq 1 - h$, RTTs in x and y have small impact on each other, i.e., x and y are non-overlapping; 3) Otherwise, overlapping. It should be noted that x and y in the last two relations cannot be noise to any other clusters. Using the pairwise relations between MSC clusters, we can identify latency patterns as follows. If two MSC clusters are overlapping, the latency pattern has overlapping modes. If two MSC clusters are non-overlapping and other clusters are noise to the non-overlapping clusters, the latency pattern has non-overlapping modes. If all other clusters are noise to one cluster, the latency pattern has a single mode.

In the algorithms above, we have three parameters λ, Δ_{th} and h. To determine λ and Δ_{th}, we establish a relation between them. Since RTTs in x and y are set to the mode of each cluster respectively before merging, the merged RTTs are comprised of alternating segments, where each segment includes RTTs of the same value and consecutive segments include RTTs of different values. We represent the merged RTTs as $\boldsymbol{r} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$, where $\boldsymbol{s}_i$ is the i-th segment, and the number of RTTs in $\boldsymbol{s_i}$ as $N(\boldsymbol{s_i})$. The relation between λ and $N(\boldsymbol{s_i})$ is as follows.

Theorem 1: Let d_x and d_y be the modes of x and y, and the merged RTTs have k segments. For $\lambda \in (0, |d_x - d_y|/2)$, we have that $\Delta_i = \frac{\lambda}{2N(\boldsymbol{s}_j)}$ if r_i is in $\boldsymbol{s}_j$ for $j \in \{1, k\}$ and that $\Delta_i = \frac{\lambda}{N(\boldsymbol{s}_j)}$ if r_i is in $\boldsymbol{s}_j$ for $j \in \{2, \ldots, k-1\}$.

Proof: The proof is provided in Appendix B. ∎

Threorem 1 tells us that the changes to RTTs depend on the number of RTTs in their segment. We set a threshold N_{th} to determine if the number of RTTs in a segment is significant. We refer segments with at least N_{th} RTTs as *significant segments*. Since we expect RTTs in significant segments have insignificant changes after TVD, we set the threshold Δ_{th} equal to λ/N_{th}. The relation between λ and Δ_{th} holds for λ between 0 and $|d_x - d_y|/2$. We thus can set λ to any value in the range. The parameters h and N_{th} both determine the percentages of different latency patterns. Higher h and N_{th} result in a larger percentage of latency patterns to be identified as having overlapping modes. As will be discussed in later sections, to guarantee the accuracy of AP-atoms, we prefer relatively large h and N_{th}. We use $h = 0.8$ and $N_{th} = 5$ in our later experiments, where the setting of N_{th} also considers the data density in our dataset as discussed in Appendix A. To avoid high-variation RTTs that are greatly inflated, we require the largest MSC cluster include at least N_{th} RTTs.

3) Clustering Clients Based on Latency Patterns: After having the latency patterns of clients, we want to cluster clients such that clients in the same cluster have similar latency patterns. The latency patterns of clients are 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 latency patterns, we divide clients having the same latency pattern into the same group and cluster clients in each group seperately. For clients with overlapping modes, since they already include individual addresses with dissimilar modes, we consider that each of them is a cluster by itself. Due to the data granularity limitation of our dataset, even if we find that addresses in some /24 clients have a distance between latencies greater than the predefined threshold, we cannot further split these clients into smaller clusters. Nonetheless, this is a limitation of our dataset, not our method. Our method can be used to split /24 clients if the measurements of individual addresses in the /24 block are available. 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 TVD algorithm discussed above. To cluster clients with a single mode, we use the quality threshold (QT) clustering algorithm [30]. 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 pre-defined threshold.

Since the algorithms to identify latency patterns and the QT algorithm are of superlinear complexity [2], [10], 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 III-A5, 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 of the 592K routable BGP prefixes provided by CAIDA.

4) 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 multiple (both past and current) periods, cumulative clusters include more clients than single-period clusters 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 third 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, where single-period clusters are obtained by clustering clients based on their latency patterns in that 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 cumulative cluster with other clients in the n-th period, 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 n-th and $(n+1)$-st periods are consolidated. We record the most recent latency pattern (i.e., the one in the $(n+1)$-st period) and correct the latency patterns of clients to their most frequent latency patterns in record if the most recent one differs from the most frequent. This is to avoid the possibility that the most recent latency pattern is misidentified. However, we do not correct the latency patterns of clients if their most frequent latency patterns are non-overlapping modes, as non-overlapping modes are 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.

5) AP-Atom Candidates: Optimal Prefix Splitting: After clustering, we obtain clusters including clients with similar latency patterns to a given server. However, these clusters only include clients with identifiable latency patterns in our dataset. 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 [28], 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 dynamic 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.

B. 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 AP-atom candidates to each server gives 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.

C. Discussions on AP-Atoms

We know how to obtain AP-atoms, but it is unclear how AP-atoms perform under specific network scenarios: middleboxes, high-variation latencies and network changes.

1) Aggregating Clients Behind Middleboxes: Middleboxes generally hide their clients' addresses from external networks. When a server communicates with clients behind a middlebox, only the public addresses (i.e., publicly routable addresses) of the middlebox are visible to the server. From the server's perspective, all measurements are from the same address. If these measurements exhibit a single latency (e.g., clients behind a NAT have similar latencies to the server), the public address is further aggregated with other clients based on the latency. In contrast, if these measurements exhibit more than one latency (e.g., clients behind a NAT have evenly distributed latencies from 100 to 200 ms to the server), the public address will be treated as a seperate aggregation unit. In AP-atoms, middleboxes that hide the private addresses from the rest of the Internet implicitly aggregate these private addresses behind them. Nonetheless, middleboxes can be further aggregated to reduce the number of aggregation units that a mapping system has to maintain. The potential improvements of AP-atoms are discussed in Section III-C4.

2) Aggregating Clients With High-Variation Latencies: Clients could have various path performance depending on their network conditions. Clients in mobile networks (e.g., cellular) are prone to experiencing more variations in latency than those in wired networks. For clients with high-variation latencies, if they are behind a middlebox, they will be aggregated by the middlebox as discussed above. In celluar networks, a majority of clients (70% in [27]) are reported behind a middlebox and have private addresses. For clients with public addresses, if they have large-variation latencies, AP-atoms will group them into small aggregation units. To understand the resilience of AP-atoms to latency variations, we conduct experiments in Section IV-C, which shows that our algorithm is able to identify latency patterns even when 30% of samples are inflated by 50% on average. As AP-atoms aggregate clients starting from /24 blocks. The worst case is that AP-atoms treat each /24 block as an aggregation unit, which increases the total number of aggregation units. This weakness of AP-atoms can be overcome if combined with active probing as discussed in Section III-C4.

3) Impact of Network Changes on AP-Atoms: When network conditions change, the performance of clients using the network might be affected, but no re-grouping of clients is needed if clients in the aggregation unit still have similar path performance after the change.[9] Only when network changes cause parts of an aggregation unit to have different performance than the others, clients in that aggregation unit should be re-grouped. Since AP-atoms group clients based on measurements, the re-grouping of clients requires new measurements be collected after the change. If a network change is long-term, AP-atoms would be updated to accommodate the change after it is detected. On the other hand, if the network change is transient, the update of AP-atoms may not be responsive to the change. The time taken to collect sufficient new measurements for re-grouping clients depends both on the portion of clients affected by the change and the traffic rates of clients. An experimental study on the responsiveness of AP-atoms to network changes is provided in Section IV-C.

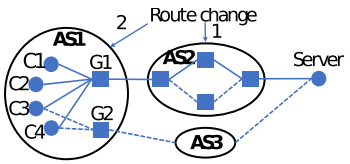

Fig. 7. Impact of network changes on client aggregation.

In Figure 7, we use route changes as an example of network changes to show how client aggregation is affected. Before the route changes, all clients in autonomous system AS1 take a path traversing gateway G1 and AS2 to the server (represented by the solid line) and have similar latencies. The first route change occurs between two routers in AS2. Since all clients still share the same path to the server, the performance of clients is changed equally and thus no client re-grouping is needed. The second route change causes clients C3 and C4 to use a new gateway G2 to the server. If the new path increases the latency to the server, clients C3 and C4 will be re-grouped into a separate aggregation unit. In contrast, if the route change does not affect the latency to the server, all clients remain in the same aggregation unit.

4) Weaknesses and Improvements of AP-Atoms: AP-atoms are obtained from passive measurements, which incurs no extra measurement overheads. Unfortunately, this also leads to several weaknesses of AP-atoms: 1) AP-atoms aggregate clients with high-variation latencies into small aggregation units, which impacts the scalability of AP-atoms; 2) AP-atoms cannot promptly react to abrupt network changes, e.g., a BGP routing change. The responsiveness of AP-atoms to network changes is determined by the traffic rates of clients; 3) since AP-atoms aggregate clients based on measurements, if the network has few measurements available, AP-atoms may not be accurate for clients in that network. These weaknesses can be overcome by using targeted active probing to complement AP-atoms [12]. Suppose we find a large IP block consisting of small aggregation units, each of which includes clients with high-variation latencies. It is possible that this large IP block includes addresses from cellular networks. We can send probes to discover if the clients in this large IP block are similar in topology (e.g., using the same gateway connecting with the Internet).[10] Similarly, we can complement passive measurements with active measurements to be more responsive in detecting network changes. The sending rate of active probes is determined by the requirements on responsiveness.

5) Extension of AP-Atoms to IPv6 Networks: AP-atoms are specifically designed for IPv4 networks. Since IPv6 has a much larger address space that is presently more sparsely allocated and carries much less traffic than IPv4 [19], the proper use of AP-atoms for IPv6 networks remains an interesting future research topic.

IV. Performance Evaluation

We use 30 days of latency measurements between clients and 10 servers in September 2015, where the locations of

[9]The reasons for this could be that clients take a new path with the same performance or that the performance of clients is changed by the same amount.

[10]It has been reported that major cellular carriers in US connect their networks with the Internet through a few ingress points [31].

Fig. 8. Locations of servers.

servers are shown in Figure 8. Due to data sparsity, as discussed in the Appendix A, we divide 30 days into 15 two-day periods and use measurements from two consecutive days to identify latency patterns. To avoid atypical latencies, we smooth the latencies by their moving average with the typical weights of moving average in TCP. 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.

A. 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.

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 9(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 9(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 $1.51\times$, $0.95\times$, and $0.54\times$ 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 less than that in the first one. The decrement is caused by the correction of 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, these clients changing 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.

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 9(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 [1], 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 9(c), the number of AP-atoms is approximately 470K for 40 servers, which is $0.8\times$ the number of BGP prefixes. For large-scale CDNs with more than 1K locations of servers [12], [23], we can select servers whose views of AP-atom candidates differ significantly and only merge AP-atom candidates of these servers.

3) Distribution of Prefix Sizes: We have compared AP-atoms with existing aggregation 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

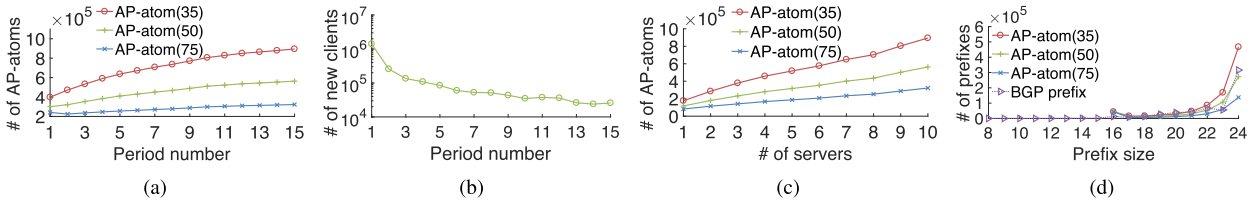

Fig. 9. Scalability of AP-atoms. (a) Scaling over time. (b) New clients over time. (c) Scaling over servers. (d) Prefix size distribution.

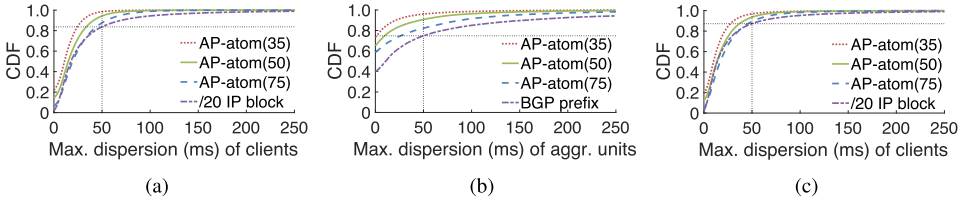

Fig. 10. Server-independence. (a) Accommodation of clients. (b) Disp. of aggr. units in the future. (c) Disp. of clients in the future.

of prefix sizes to understand the advantages of AP-atoms over existing aggregations. Figure 9(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.

B. 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 II-A 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.

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 10(a) shows the maximum dispersion of clients calculated using latencies from the first to the 15hbox-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.

2) Dispersion and Latency Prediction: At the end of each period, we update AP-atoms and use them for the next period. AP-atoms used in the $(n+1)$-st period are AP-atoms updated when the n-th period ends. From Figure 9(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 sufficient 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 10(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 10(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 10(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 $0.46\times$ the number of /20 IP blocks, AP-atoms(75) can reduce this number to 10%. With a scale that is $1.3\times$ the number of /20 IP blocks, AP-atoms(35) reduces this number to 3.5%.

C. Tradeoff Between Tolerance and Responsiveness

Clients could have high-variation latencies to a server if their paths are greatly inflated due to network events, e.g., queuing and route changes. Our latency identification algorithm can tolerate a certain level of inflation and still accurately identify the underlying latency pattern. A larger set of measurements is always preferred for latency identification, but collecting measurements takes time, influencing the responsiveness of AP-atoms to network changes. We want to study the tradeoff between tolerance and responsiveness.

To test our algorithm, we create synthetic latency samples with different levels of inflations, controlled by the probability of latency samples being inflated (denoted as r). Suppose the base latency without inflation is denoted as l. In our experiments, each sample is chosen to be inflated with a probability of r. If a sample is inflated, we first determine the inflation that is uniformly distributed between 0 and l,[11] and then add the inflation to the base latency; otherwise, the sample is equal to the base latency. We continue generating new samples until the desired number of samples (denoted as n) is reached. We then apply our algorithm on these samples to identify the base latency. If a single latency is found in these samples, we consider that our algorithm succeeds in identifying the base latency and calculate the *identification error* (i.e., the difference between the identified base latency and the actual base latency).

In our experiments, we set the base latency to 100 ms such that the maximum inflation is 100 ms, large enough to simulate variable network conditions. To avoid randomness, we repeat the same experiment 1000 times and use the average results. Figure 11 shows the average success and error ratios under different r's and n's, where the *success ratio* is the percentage of repeated experiments in which our algorithm succeeds in identifying the base latency and the *error ratio* is the ratio of the *identification error* to the base latency. Figure 11(a)

[11]We use a simple uniform distribution to model latency inflation. The maximum inflation is set to l, proportional to the base latency, as a path with larger latency is likely to traverse more links and experience larger inflation.

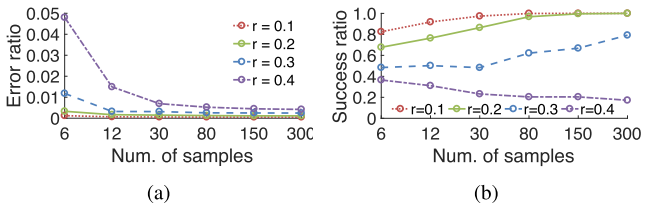

Fig. 11. Tolerance to latency inflation. (a) Error ratio. (b) Success ratio.

shows that the identified base latency is accurate with an error less than 5% for all r's, but the cost of having accurate identification is that the success ratio is relatively low when the number of samples is small. Using a larger number of samples can improve the success ratio to near 100% when $r = 0.1$ or 0.2, but this fails when r increases to 0.4. This implies that the limit to the tolerance of our algorithm to latency inflation is reached when 30% of samples are inflated 50 ms on average.

The success ratio determines the percentage of clients whose latencies can be identified. A better tolerance to inflation leads to a higher success ratio. We want a better tolerance in order to have more clients with identifiable latencies. However, collecting a larger number of measurements generally takes a longer time, resulting in slow response to network changes. Considering both tolerance and responsiveness, using 12 samples for latency identification is a reasonable choice.[12] When no base latency is identified at 12 samples, we can either consider the clients have multiple latencies and regroup them or wait for more samples to be collected. The actual time taken to collect 12 samples depends on the traffic rates of clients. We can complement passive measurements with active probes to improve the responsiveness to network changes.

Compared to AP-atoms, other aggregation methods are not adaptive to network changes. Geolocations and LDNSs of clients are static. Although BGP prefixes may be changed, as BGP prefixes are designed for Internet routing, changing BGP prefixes would cause many BGP issues like convergence and incur communication overheads. As found in [25], except for new networks, most new BGP prefixes are due to BGP misconfigurations. Moreover, BGP prefixes are managed by tens of thousands of service providers separately. Cooperation from these service providers is needed to change BGP prefixes.

V. 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 [3]. 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 [29]. By considering BGP prefixes as a natural group of Internet clients, 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

[12]In previous experiments, we divide time into fixed-length periods and use measurements from the same periods for comparison purpose.

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 [22]. In iPlane, clients are aggregated into BGP prefixes to construct a topological map for the Internet [16]. 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 [26]. The geographic locality of BGP prefixes has also been studied [24]. Recently, Lee and Spring [31] 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 [11], [26], 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 [8], [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 [12]. However, end-user mapping requires the mapping system estimate the performance from servers directly to client aggregations [11]. Thus, a proper aggregation is crucial to both the scalability and accuracy of the mapping system. Using a heuristic method, Chen *et al.* [12] 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. Gharaibeh *et al.* [21] studied the similarity between individual addresses in /24 prefixes in terms of geographic co-locality. Lee and Spring [31] studied the similarity between individual addresses in /24 prefixes in terms of topological proximity. Both works confirm that /24 prefixes are not necessarily the smallest aggregation units.

VI. 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 $2\times$ and reduce the 98-th percentile difference in clients' latencies by almost 100ms.

For future work, we want to further improve the scalability of AP-atoms and use AP-atoms for performance prediction, which could significantly reduce active measurement loads and enable intelligent mechanism to detect network congestions.

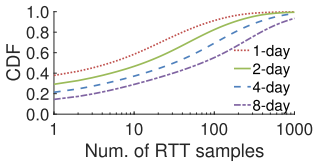

Fig. 12. Data density.

Appendix A
Data Description

Our dataset is provided by a commercial global load balancing service provider, collecting over 1 billion measurements per day as part of their normal operations. Each time when a client browses an instrumented web hosted by a participating provider, a Javascript script is downloaded and executed to download test objects. The test objects are small enough to fit into a single TCP packet. The time to first byte is recorded as the latency to the participating provider. This project is called Radar and is described in detail in [7]. Measurements are conducted when the browser is idle such that user experience is not affected. Since each visit to the web pages will trigger only a few measurements, the measurements from single clients are sparse over time. Our dataset includes measurements from both wireless (Wi-Fi only) and wired networks, where the percentage of measurements from wireless networks is 12.8%. Since measurements in our dataset are triggered by user activities, our dataset resembles passive measurements that global load balancers would obtain from user traffic. The anonymization of addresses in our dataset does not limit the use of our dataset as explained in Section III-A1.

Our dataset covers 240 countries (280 in total) listed in ISO 3166 and 34% of Autonomous Systems (ASs) in CAIDA's database [5], including all Tier-1 ASs, with 36% of the remainder being Tier-2 ASs (directly connected to Tier-1 ASs). Clients in the data come from 4.2M /24s and have traffic with 160 service provider sites. However, most providers do not have enough traffic with clients for analysis purposes. We only use the top 10 servers having the most traffic with clients in our experiments. Figure 12 shows the distribution of the number of RTT samples between clients and servers in period lengths of 1 to 8 days. The measurements of individual addresses in the same /24 block are aggregated and are considered from the same single client. 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.

Appendix B
Proof of Theorem 1

Without loss of generality, we assume $d_x < d_y$. 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 in distance. Similarly, when the optimal solution is reached, for r_i's in the same segment, we must not have oscilating Δ_i's, e.g., $\Delta_i > \Delta_{i-1}$ and $\Delta_i > \Delta_{i+1}$. For such Δ_i's, a smaller cost can

be achieved by setting $\Delta_i = (\Delta_{i-1} + \Delta_{i+1})/2$. Thus, for all r_i's in the same segment, we have a monotonic sequence of Δ_i's. Without loss of generality, assuming that the sequence of Δ_i's in the same segment is non-decreasing, we have $|z_i - z_{i-1}| = |\Delta_i - \Delta_{i-1}| = \Delta_i - \Delta_{i-1}$. By denoting $d_y - d_x$ as Δd and $z_i - z_{i-1}$ as Δz_i, we can rewrite the cost function as

$$f = \sum_{i=1}^{m} \Delta_i^2 + \lambda \left(\Delta d - \Delta_1 - \Delta_{N(\boldsymbol{s}_1)+1} + \sum_{i=N(\boldsymbol{s}_1)+2}^{m} |\Delta z_i| \right).$$

When f is minimized, we have that for each r_i where $i \in \{2, \ldots, N(s_1)\}$, $\partial f / \partial \Delta_i = 2\Delta_i = 0$. However, since the sequence of Δ_i is non-decreasing, we have that f is minimized when $\Delta_i = \Delta_1$ for $i \in \{2, \ldots, N(s_1)\}$. This implies that the cost is minimized when all r_i's in the same segment have the same change. Let Δ_{s_i} be the change of RTTs on the i-th segment. We have

$$f = \sum_{i=1}^{k} N(\boldsymbol{s}_i) \Delta_{\boldsymbol{s}_i}^2 + \lambda \left((k-1)\Delta d - \sum_{i=1}^{k} \Delta_{\boldsymbol{s}_i} - \sum_{i=2}^{k-1} \Delta_{\boldsymbol{s}_i} \right).$$

Taking the derivatives of f with respect to Δ_{s_i}, we have that the optimal solution is achieved when $\Delta_{\boldsymbol{s}_i} = \frac{\lambda}{2N(s_i)}$ for $i \in \{1, k\}$ and $\Delta_{\boldsymbol{s}_i} = \frac{\lambda}{N(\boldsymbol{s}_i)}$ for $i \in \{2, \ldots, k-1\}$. For the similar reason as Δ_i's, we must not have oscilating $\Delta_{\boldsymbol{s}_i}$'s across segements, i.e., $\frac{\lambda}{N(\boldsymbol{s}_i)} + \frac{\lambda}{N(\boldsymbol{s}_j)} \leq \Delta d$ is true for any $N(\boldsymbol{s}_j)$ and $N(\boldsymbol{s}_j)$ when $i \neq j$. We thus have $\lambda \leq \Delta d/2$.

REFERENCES

[1] *USC CDN Coverage*. Accessed: May 6, 2017. [Online]. Available: http://usc-nsl.github.io/cdn-coverage

[2] A. Danalis, C. McCurdy, and J. S. Vetter, "Efficient quality threshold clustering for parallel architectures," in *Proc. Parallel Distrib. Process. Symp.*, May 2012, pp. 1068–1079.

[3] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," in *Proc. ACM SIGCOMM CCR*, 2015, pp. 52–66.

[4] CAIDA. (May 2015). *The CAIDA AS Relationships Dataset*. [Online]. Available: http://www.caida.org/data/as-relationships/

[5] CAIDA. *Routeviews Prefix to AS Mappings Dataset for IPv4 and IPv6*. Accessed: Sep. 1, 2015. [Online]. Available: http://www.caida.org/data/routing/routeviews-prefix2as.xml

[6] C. Contavalli, W. van der Gaast, D. Lawrence, and W. Kumari, *Client Subnet in DNS Queries*, document RFC 7871, May 2016.

[7] Cedexis. *Radar Crowd Sourcing*. Accessed: Feb. 25, 2018. [Online]. Available: https://www.cedexis.com/technology/

[8] C. Huang, I. Batanov, and J. Li, "A practical solution to the client-LDNS mismatch problem," in *ACM SIGCOMM CCR*, 2012, pp. 35–41.

[9] Citrix. *NetScaler Global Server Load Balancer*. [Online]. Available: https://docs.citrix.com/en-us/netscaler/12/global-server-load-balancing.html

[10] D. Freedman and P. Kisilev, "Fast mean shift by compact density representation," in *Proc. CVPR*, Jun. 2009, pp. 1818–1825.

[11] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," in *Proc. SIGOPS*, 2010, pp. 2–19.

[12] F. Chen, R. K. Sitaraman, and M. Torres, "End-user mapping: Next generation request routing for content delivery," in *Proc. SIGCOMM*, 2015, pp. 167–181.

[13] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. SIGCOMM*, 2004, pp. 15–26.

[14] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," in *Proc. SIGCOMM*, 2002, pp. 75–88.

[15] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, "UniDrive: Synergize multiple consumer cloud storage services," in *Proc. 16th Annu. Middleware Conf.*, 2015, pp. 137–148.

[16] H. V. Madhyastha *et al.*, "iPlane: An information plane for distributed services," in *Proc. OSDI*, 2006, pp. 367–380.

[17] IP2Location. *Geolocate IP Address Location Using IP2Location*. Accessed: Oct. 18, 2016. [Online]. Available: http://www.ip2location.com

[18] I. Selesnick. (2014). *Total Variation Denoising (An MM Algorithm)*. [Online]. Available: https://goo.gl/rjBpG

[19] J. Czyz *et al.*, "Measuring IPv6 adoption," in *Proc. SIGCOMM*, 2014, pp. 87–98.

[20] J. S. Otto, M. A. Sánchez, J. P. Rula, and F. E. Bustamante, "Content delivery and the natural evolution of DNS," in *Proc. IMC*, 2012, pp. 1–14.

[21] M. Gharaibeh, H. Zhang, C. Papadopoulos, and J. Heidemann, "Assessing co-locality of IP blocks," in *Proc. IEEE Global Internet Symp.*, Apr. 2016, pp. 503–508.

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

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

[24] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan, "Geographic locality of IP prefixes," in *Proc. USENIX ATC*, 2005, pp. 153–158.

[25] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. ACM SIGCOMM CCR*, 2002, pp. 3–16.

[26] R. Krishnan *et al.*, "Moving beyond end-to-end path information to optimize CDN performance," in *Proc. IMC*, 2009, pp. 190–201.

[27] S. Triukose, S. Ardon, A. Mahanti, and A. Seth, "Geolocating IP addresses in cellular data networks," in *Proc. PAM*, 2012, pp. 158–167.

[28] T. Hayashi and T. Miyazaki, "High-speed table lookup engine for IPv6 longest prefix match," in *Proc. IEEE GLOBECOM*, Dec. 1999, pp. 1576–1581.

[29] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for Internet hosts," in *Proc. SIGCOMM*, 2001, pp. 173–185.

[30] X. Jin and J. Han, "Quality threshold clustering," in *Encyclopedia of Machine Learning*. Boston, MA, USA: Springer, 2011, p. 820.

[31] Y. Lee and N. Spring, "Identifying and aggregating homogeneous IPv4/24 blocks with hobbit," in *Proc. IMC*, 2016, pp. 151–165.

Yibo Pi received the B.S. and M.S. degrees in electrical and computer engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with the University of Michigan, Ann Arbor. His research interests include Internet measurement.

Sugih Jamin is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. His research interests include measurement-based applications.

Peter Danzig was an Associate Professor with the University of Southern California, Los Angeles. He is currently with Panier Analytics, Menlo Park, CA, USA.

Jacob Shaha received the B.S. degree in computer engineering from the University of Utah in 2006 and the M.S. degrees in applied statistics and electrical engineering systems from Pennsylvania State University and the University of Michigan in 2014 and 2016, respectively. He is currently an Instructor with the United States Military Academy, West Point.