A Global Name Service for a Highly Mobile Internet

Xiaozheng Tie
University of Massachusetts Amherst
xztie@cs.umass.edu

Arun Venkataramani
University of Massachusetts Amherst
arun@cs.umass.edu

ABSTRACT

Mobile devices dominate the Internet today, however the Internet continues to operate in a manner similar to its early days with poor infrastructure support for mobility and multihoming of devices, content, and networks. Our position is that in order to address this problem, a key challenge that must be addressed is the design of a massively scalable distributed name resolution infrastructure that rapidly resolves identities to network locations under high mobility.

Our primary contribution is the design, implementation, and evaluation of Auspice, a next-generation global name resolution service that addresses this challenge. The key insight underlying Auspice is a resolver replica placement engine that automates the placement of resolvers for names in a locality- and load-aware manner. Auspice employs a heuristic placement algorithm that determines the number and locations of resolver replicas so as to reduce user-perceived response times and update cost while ensuring that no replica location becomes a load hotspot. Our experiments over PlanetLab, a local cluster, as well as large-scale trace-driven experiments show that Auspice outperforms both commercial managed DNS services and proposed DHT-based placement alternatives to DNS by up to an order of magnitude.

Keywords

Network architecture and design, distributed networks, network communication

1. INTRODUCTION

"Mobile" has long arrived, but the Internet remains unmoved. Today, there are almost 6 billion cellphones, over a billion of which are smartphones [14]. The number of smartphones alone now exceeds the number of tethered hosts on the Internet, and a recent Cisco whitepaper [3] suggests that the total traffic generated by mobiles now exceeds that of tethered hosts. However, the current Internet continues to operate as it did when dominated by tethered hosts, simply ignoring the fact that users, content, and networks (e.g., an intra-vehicular network) are frequently mobile.

A mobile user might reasonably expect that a voice-over-IP call she initiated through one WiFi network would continue uninterrupted if she switched to a different WiFi or cellular network; or expect a file transfer she initiated at home on her laptop to resume when she opens it at work in a disruption-tolerant manner. Today, one can not easily initiate communication with a smartphone (even when it receives a publicly visible IP address) because there is no globally available infrastructure support for it. Of course, application developers can design around these limitations, as do applications like Skype and Dropbox for the above scenarios. However, the consequence is that we are paying an unknowable price in terms of stymied application innovation and growth by forcing developers to redundantly develop common-case functionality, and forcing communication initiation to be mostly unidirectional.

Many before us have lamented in a similar vein about the Internet architecture's poor support not only for mobility but also for multihoming [15] and content retrieval [17, 12]. A common criticism is the Internet's so-called conflation of identity and location. The Internet uses an IP address both to represent the identity of an interface as well as its network location, which is problematic for mobility (same identity, changing locations) and multihoming (single identity, multiple locations) of devices, services, or content. Applications today are forced to know and care about changing IP addresses as the transport and network layers only provide a primitive to establish connections between IP addresses, not application-friendly names. It is common accepted wisdom that a cleaner separation of identity and location is instrumental to fixing these problems.

However, the Internet does separate identities (human-readable domain-names) from network locations (IP addresses) through DNS. Most high-level programming languages also provide syntactic sugar to "connect" to names remaining oblivious to IP addresses; name owners can and do employ DNS, managed DNS services, or CDNs to return the best-positioned network location corresponding to multi-homed names. Techniques from a long line of work on connection migration can be employed if needed to seamlessly handle mid-session mobility in a bilateral manner, especially in mobile apps where the application developer can easily modify both ends. But a key missing element in this package today is a distributed resolution infrastructure that can scale to orders of magnitude higher update rates than envisioned when DNS was created. To appreciate our envisioned scale, consider 10 billion mobile identifiers moving across a 100 different networks per day (or an

aggregate rate of roughly 1M/sec). DNS's heavy reliance on TTL-based caching, a key strength recognized by its creators, researchers, and operators alike, also poses a significant handicap by increasing update propagation delays. It is not uncommon for DNS update propagation to take on the order of days, resulting in long service outage times when online services have to be moved in an unanticipated manner, prompting cries for help on operator forums [9].

To address this problem, we present Auspice, a global name resolution service that rapidly resolves identities to locations irrespective of how exactly identities and locations are (separately) represented. The key strength of Auspice is an engine for *automated* placement of replicas of resolvers for a name in a locality- and load-aware manner. Auspice achieves locality-awareness by inferring pockets of high demand for a name so as to create replicas of resolvers for that name close to them. Auspice achieves low response time, low update cost, and load balance across all resolver locations using a placement optimization algorithm that (1) controls the number of resolver replicas based on the observed lookup and update rates, and (2) determines where to place replicas based both on the inferred pockets of demand and the aggregate load at locations near those pockets.

We have implemented a prototype of Auspice and evaluated it through several different platforms including PlanetLab, a local cluster, and large-scale trace-driven evaluation. Our results show that Auspice significantly reduces user-perceived response time and update cost, and outperforms commercial DNS services and DHT-based placement alternatives to DNS by an order of magnitude.

2. AUSPICE DESIGN

In this section, we present the design and implementation of Auspice, a global name resolution service that resolves domain names to locations. At the heart of Auspice is a resolver placement engine that *automatically* places resolver replicas in a locality- and load-aware manner so as achieve low response time, low update cost, and load balance. Below I first show that today's domain names exhibit high geo-locality characteristics, and then use this insight to drive the design of Auspice.

2.1 Geographic locality

If lookup patterns show geo-locality, placing a small number of resolvers in a locality-aware manner can achieve low response time at a low cost. To assess the extent of geo-locality in DNS names today, we conduct an analysis of Alexa's dataset [1] that provides statistics about page views (refer to §3 for more details of this dataset) that we expect to be correlated with the number of name lookups for the corresponding domain name.

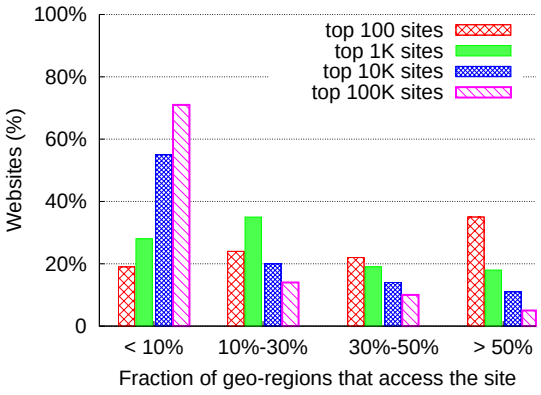

Figure 1: Locality in website popularity: 20% websites are accessed in less than 10% of geographic regions.

Figure 1 shows the geo-locality in website popularity: it plots the percentage of websites that are accessed in different fraction of geographical land regions[1]. 20% websites are accessed in less than 10% of the geo regions for the top 100 and 1K sites and the percentage value goes up to 70% for the top 100K sites. Further, we find that for two of the top 10 sites (baidu.com and qq.com), 95% of the demand comes from a single country, China, while for a third, amazon.com, 75% of the demand originates from the US alone. This observed high locality of even popular names today suggests that locality-aware replica placement can significantly reduce lookup response time at a low update cost. We expect traffic originating from or destined to mobile names to exhibit even more locality, especially with the increasing growth of location-aware mobile services. Thus, locality-awareness is an important design goal in Auspice.

2.2 Auspice design

Auspice is designed as follows. Each name is associated with a fixed number, M, of *replica-controllers* and a variable number of *active replicas* of the corresponding resolver. The locations of the replica-controllers is fixed and computed using M well-known consistent hash functions. The replica-controllers are responsible only for determining the number and locations of active replicas, and the active replicas are responsible for maintaining resource records for names and responding to requests from end-users. The replica-controller implement a replicated state machine to maintain a consistent view of the current set of active replicas despite failures.

The computation of the active replica locations for each name proceeds in epochs as follows. At bootstrap time, the active replicas are chosen to be physically at the same locations as the corresponding replica-

[1]To measure it, we divide the world's land region into 1000 grids and count access from any point in a grid as access from that grid.

controllers. In each epoch, the replica-controllers obtain from each active replica a summarized load report that contains the request rates for that name from different *regions* as seen by that replica. Regions could either be access networks or geographic regions that partition users into non-overlapping groups so as to capture locality. Thus, each active replica's load report consists of a spatial vector of request rates as seen by that replica. The replica-controllers aggregate these load reports to obtain a concise spatial distribution of all requests for the name.

2.2.1 Mapping algorithm

In each epoch, the replica-controllers use a *mapping algorithm* that takes as input the aggregated load reports and capacity constraints at all resolvers locations to determine the number and locations of active resolver replicas for each name. The mapping algorithm is a simple heuristic algorithm which can be run independently by each replication controller. The algorithm computes the number of replicas using the lookup to update ratio of a name in order to limit the update cost to within a small factor of the lookup cost. The number of replicas is always kept more than the minimum number needed to ensure fault tolerance. The location of these replicas are decided partly based on the locality of demand, while the rest of the replicas are placed randomly in order to achieve a good tradeoff between locality-awareness and load-balance.

Specifically, the mapping algorithm computes the number of replicas for a name as $(M + \beta R_i/W_i)$, where R_i and W_i are the lookup and update rates of name i, M is the minimum number of replicas needed to ensure fault tolerance, and β is a replication control parameter. The mapping algorithm dynamically adjusts β to control the number of replicas and ensure low update cost at various loads.

The replica-controllers compute β in each epoch so as to ensure that the aggregate load in the system corresponds to a predetermined threshold utilization level μ (e.g., 75%). For simplicity of exposition, suppose a lookup and update operations impose the same load, and the total capacity across all name resolvers (in (lookups + updates) /sec) is C. Then, β is set so that

$$\mu C = \sum_i R_i + \sum_i (M + \beta \frac{R_i}{W_i}) W_i \qquad (1)$$

where the right hand side represents the total lookup and update load summed across all names. The first term in the summation above is the total lookup load and the second is the total update load. Thus,

$$\beta = \frac{C\mu - M \sum_i W_i}{\sum_i R_i} - 1 \qquad (2)$$

Having computed β as above, Auspice computes the locations of replicas for name i as follows. Out of the $M + \beta R_i/W_i$ total replicas, a fixed number, K, of replicas are chosen according to locality, i.e., the replica-controllers use the spatial vector of load reports to select K resolver locations that are respectively the *closest* to the top K regions sorted by demand for name i. The remaining $M + \beta R_i/W_i - K$ replicas are chosen randomly without repetition.

The top-K "closest" replicas above are chosen as the closest with respect to round-trip latency plus load-induced latency between regions and name servers. An earlier design simply chose the top-K according to round-trip locality alone; however, we found that adding load-induced latencies in this step in addition to choosing the remaining replicas randomly ensures better load balance and results in lower overall user-perceived response times.

2.2.2 Routing client requests

The list of all name resolver locations is well known and can be obtained by contacting any resolver. End-hosts can either directly send lookups to the name service or channel them through a local name server like today. Local name servers obtain the set of active resolver replicas from the replica-controllers and send lookups to the closest active replica. To decide which active replica is closest, local name servers maintain latency estimates to name resolvers that reflect both network latency and resolver-load induced latency (or resolver latency).

A local name server must maintain these latency estimates efficiently for up to a thousands of resolvers. Network latency estimates change slowly and therefore every local name server maintains round-trip latency estimates to all resolvers using infrequent measurements. Resolver latency changes frequently due to dynamic loads, therefore the local name server maintains current resolver latency updated only for a smaller number of frequently contacted resolvers.

2.3 Implementation

My implementation consists of two stand-alone modules, name resolver and local name server, described below.

Name resolver: By default, Auspice uses the following parameter values. The number of replica-controllers and the minimum number of active replicas for fault-tolerance are set to $M = 3$. Auspice chooses the top $K = 5$ replicas in a locality-aware manner and the rest of the replicas are chosen randomly. The replication controller computes a new placement of replicas once every 5 minutes and resolvers submit summary reports to the replication controllers once every 5 minutes.

Local name server: Local name server maintains a TTL-based cache of name records. On a cache miss, local name server queries up to three active replicas to get a response. A timeout value is set for every lookup sent and the lookup is sent to the next closest active replica

on a timeout. The timeout value is decided adaptively based on latency of past lookups.

3. EVALUATION

We implemented and deployed Auspice on two different platforms PlanetLab [8] and a local cluster. We extensively evaluate Auspice using our prototype deployment as well as large-scale trace-driven experiments. Our evaluation shows the following:

- *Low response time:* Auspice's locality-aware placement performs close to an optimal placement algorithm, and achieves 5× lower response time than a DHT-based algorithm.
- *Low update cost:* Auspice's load-awareness reduces updates costs by nearly an order of magnitude over a replicate-everywhere algorithm and yet achieves nearly identical response time.
- *Load balance:* Over a wide range of loads, Auspice has comparable load balance performance compared to DHT- and random-based placement algorithms.
- *Comparison to managed DNS:* Auspice achieves DNS lookup response time comparable to several leading managed DNS providers today even with only one third name resolvers.

3.1 Experimental setup

Our workload consists of name lookups and updates from clients spread across the globe. The workload has two types of names: (1) *Service names* correspond to names for web services, e.g., amazon.com (2) *Device names* are names for mobile devices. Lookups for service names are generated based on the Alexa dataset [1], which reports the relative popularity of top Internet websites and the geo-distribution of their popularity at city-level granularity. Each city is mapped to the geographically closest local name server. Updates for service names are a tiny fraction (0.01%) of lookups.

We generate a synthetic workload for device names. In most experiments, the total number of lookups is equal to that of updates for device names. The geo-distribution of lookups for a device name is as follows. 90% of lookups are generated from between 1% and 5% of local name servers which are geographically close to each other and the remaining 10% lookups are generated from randomly chosen local name servers.

We compare Auspice against four alternative name service placement algorithms: (a) *Optimal*, which formulates the placement problem as a mixed-integer optimization problem (described in [21]). Optimal decides both resolver placement and client request routing to minimize the total network and resolver latency. (b) *CoDoNS* [18], which replicates names and places resolvers over Pastry DHT. (c) *Static-3*, which replicates each name at three random locations. It has the smallest update cost and it evenly distributes names among resolvers. (d) *Replicate-All*, which replicates all names at all locations. In doing so, it naively optimizes response time without considering the cost of updates.

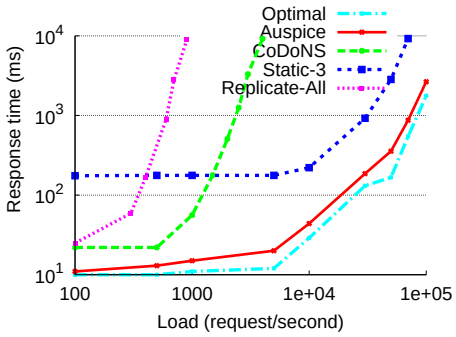

Figure 2: Auspice achieves 5× lower response time than CoDoNS.

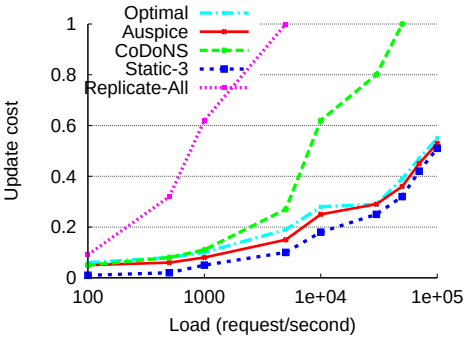

Figure 3: Auspice reduces update cost by an order of magnitude over Replicate-All.

3.2 Response time

Figure 2 shows the median response time across names for different placement algorithms, as we increase the total lookup and update loads from 100 requests per second to 100,000 requests per second. Each dot in the figure is the median value of the mean response time for all names, where the mean response time for a name is the average response time of all lookup requests for the name.

Figure 2 shows that Auspice performs close to Optimal, and achieves up to 5× lower response time over CoDoNS, a state-of-the-art DHT based placement algorithm. This indicates that Auspice's locality-aware placement effectively places resolver replicas close to pockets of demands. In comparison, CoDoNS has high response time because its DHT-based routing doesn't place replicas according to high locality of demands.

3.3 Update cost

Figure 3 shows the median update cost at varying load. Each dot is the median value of the mean update cost for all names, and the mean update cost for a name is the product of the number of resolvers and the update rate for that name.

The figure shows that Auspice reduces update cost by

	Fairness index
Auspice	0.95 (0.08)
CoDoNS	0.96 (0.04)
Static-3	0.98 (0.04)
Replicate-All	1.00 (0.00)

Table 1:

an order of magnitude over Replicate-All, and it has update cost as low as Static-3, a placement algorithm that incurs the lowest update cost by creating the the least amount of replicas. This attests the effectiveness of Auspice's load-aware placement that dynamically reduces the number of replicas for each name when it observes a high network load. In comparison, Replicate-All incurs the highest update cost because it replicates names at all name resolvers.

3.4 Load balance

Table 1 shows the load balance performance for different algorithms. We measure this metric by computing Jain's fairness index [7]. When there are n name resolvers with load x_1 through x_n, the fairness index is computed as

$$\frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$$

Table 1 shows the mean fairness index across varying loads for different algorithms. Auspice has fairness index comparable to the other algorithms.

3.5 Comparison to managed DNS

Response time →	**Median**	**Mean**	**95-%ile**
Auspice	45	74	246
Managed DNS	40	72	259

Table 2: Auspice has response times comparable to managed DNS provider with less than one-third replicas.

Managed DNS providers such as UltraDNS [10], DynDNS [6], and DNSMadeEasy [4] offer a geo-replicated authoritative DNS service similar to Auspice. In this subsection, we compare the lookup response times between Auspice and a leading managed DNS provider.

Our workload consists of lookups for domain names serviced by the provider. We identify 316 domain names among the top-10K Alexa websites [1] serviced by this provider. This provider replicates each name record at 16 locations. We limit the maximum number of replicas to 5 in order to provide an even comparison between Auspice and the provider.

Table 2 shows the median, mean and 95th percentile lookup response times across names for Auspice and the managed DNS provider. Auspice performs within 11% of the managed DNS provider for all metrics. While Auspice creates less than one-third replicas as the managed DNS provider, it places them considering the geolocality of demand. Due to its placement algorithm, Auspice achieves similar response time with smaller cost of updates.

4. RELATED WORK

Our work on Auspice draws on lessons learned from an enormous body of prior work on distributed systems for name resolution as well as more general services. Compared to this prior work, the novel contribution in Auspice is an engine to *automate* wide-area replica placement in a locality- and load-aware manner, as explained in detail below.

Naming: Until the early 80s, the Internet relied on a system called HOSTS.TXT for name resolution, which was simply a centrally maintained text file and distributed to all hosts. The current Internet's distributed Domain Name System [5] arose in response to the rapidly increasing size of the file and the cost of distributing it. Mockapetris and Dunlap point to TTL-limited caching to reduce load as well as response times as a key strength, noting that "the XEROX system (Grapevine [19]) was then ... the most sophisticated name service in existence, but it was not clear that its heavy use of replication, light use of caching ... were appropriate". We have since come a full circle, falling back on heavy use of active replication in Auspice in order to address the challenges of mobility, a concern that was not particularly pressing in the eighties.

Server selection: A number of prior systems have addressed the server selection problem where data or services are replicated across a wide-area network. OASIS [13] maps users based on IP addresses to the best server based on latency and server load. DONAR [22] enables an expressive API for content providers to specify performance or cost optimization objectives while meeting load balancing constraints. These systems as well as commercial CDNs and cloud hosting providers [2] share our goals of proximate server selection and load balance given a fixed placement of server replicas. In comparison, our approach additionally considers replica placement itself as a degree of freedom in achieving latency or load balancing objectives.

Placement: Volley [11] optimizes the placement of dynamic data objects based on the geographic distribution of accesses to the object and is similar in spirit to Auspice in that respect. However, Volley implicitly assumes a single replica for each object and therefore does not have to worry about high update rates or coordination overhead for replica consistency. As shown in §3, creating multiple replicas of objects or services can significantly reduce user-perceived response times while also enhancing opportunities to balance load provided update cost is taken into account.

CDNs dynamically cache copies of static content near points of demand. In comparison, Auspice makes active replicas of dynamic objects (or resolver services) in a planned manner. Planned placement is typically considered unnecessary for static content distribution as simple caching and replacement schemes suffice to capture most of the benefit[16, 20]. In this respect, Auspice is comparable to edge services offered by commercial CDNs. However, the geo-distributed locations of edge services as well as cloud-hosted services today are chosen manually and infrequently updated. In comparison, Auspice automates geo-distributed placement of service replicas, but the service, name resolution, is simpler compared to black-box cloud-hosted services.

5. CONCLUSIONS

In this paper, we presented the design, implementation, and evaluation of Auspice, a massively scalable, distributed, global name service for an Internet where high mobility is the norm. The name service can resolve flexible identifiers (human-readable names, self-certifying identifiers, or arbitrary strings) to network locations that can also be defined in a flexible manner. At the heart of Auspice is a resolver replica placement engine that automates placement of resolver replicas for each name in a locality- and load-aware manner so as to ensure low user-perceived response time, low update cost, and load balance. Our evaluation shows that Auspice's placement algorithms significantly outperforms both commercial managed DNS services employing simplistic replication strategies as well as previously proposed DHT-based replication alternatives.

6. RESEARCH PHILOSOPHY

My research philosophy is strongly based on building real systems and applying theoretical analysis to better understand these systems. Building and stressing systems in real environments enables my research to identify practical problems and shed light on new insights that are not easily uncovered using small prototype systems or simulation. Analyzing problems using theoretical techniques further strengthens my research approach and generalizes my solution to fundamental principles applicable beyond the specific system. I am also a strong adherent of cross-disciplinary research. I have drawn ideas from other fields including statistics, networking theory and social networks. For instance, I quantify the performance gap between Auspice and the optimal strategy by formulating service placement as a mixed-integer optimization problem that is computationally hard.

7. REFERENCES

[1] Alexa web information service. `http://aws.amazon.com/awis/`.

[2] Amazon elastic load balancing. `http://aws.amazon.com/elasticloadbalancing/`.

[3] Cisco mobile traffic report. `http://blogs.cisco.com/tag/mobile-data-traffic/`.

[4] Dns made easy. `http://www.dnsmadeeasy.com`.

[5] Domain name system. `http://en.wikipedia.org/wiki/Domain_Name_System`.

[6] Dyn dns. `http://dyn.com/`.

[7] Jain's fairness index. `http://en.wikipedia.org/wiki/Fairness_measure/`.

[8] Planetlab. `https://www.planet-lab.org/`.

[9] Serverfault. `http://serverfault.com/questions/41018/dns-any-way-to-force-a-nameserver-to-update-the-reco`

[10] Ultra dns. `http://www.neustar.biz/`.

[11] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: automated data placement for geo-distributed cloud services. NSDI'10.

[12] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the internet. In *SIGCOMM*, 2004.

[13] M. J. Freedman, K. Lakshminarayanan, and D. MaziŔres. Oasis: Anycast for any service, 2006.

[14] Gartner. Mobile Connections Will Reach 5.6 Billion in 2011, 2011. `http://www.gartner.com/it/page.jsp?id=1759714`.

[15] P. Jokela, P. Nikander, J. Melen, J. Ylitalo, and J. Wall. Host identity protocol, extended abstract. In *Wireless World Research Forum*, 2004.

[16] M. Karlsson and M. Mahalingam. Do we need replica placement algorithms in content delivery networks? In *WCW*, 2002.

[17] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. SIGCOMM '07.

[18] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. SIGCOMM '04.

[19] M. D. Schroeder, A. D. Birrell, and R. M. Needham. Experience with grapevine: the growth of a distributed system. *ACM Trans. Comput. Syst.*

[20] A. Sharma, A. Venkataramani, and R. Sitaraman. Distributing Content Simplifies ISP Traffic Engineering. *Technical Report*, 2012. `http://people.cs.umass.edu/~abhigyan/NCDN.pdf`.

[21] X. Tie, A. Sharma, and A. Venkataramani. A global name service for a highly mobile internet. Technical report, UMASS, 2013.

[22] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. Donar: decentralized server selection for cloud services. SIGCOMM, 2010.