Design Requirements of a Global Name Service for a Mobility-Centric, Trustworthy Internetwork

Arun Venkataramani*, Abhigyan Sharma*, Xiaozheng Tie*, Hardeep Uppal*,
David Westbrook*, Jim Kurose*, Dipankar Raychaudhuri+
*University of Massachusetts Amherst, +Rutgers University
{arun, abhigyan, xztie, hardeep, westy, kurose}@cs.umass.edu, ray@winlab.rutgers.edu

Abstract

The Internet's tremendous success as well as our maturing realization of its architectural shortcomings have attracted significant research attention towards clean-slate re-designs in recent times. A number of these shortcomings can be traced back to *naming*. The current Internet uses IP addresses to conflate identity and network location, which results in poor support for mobility and multihoming; vulnerability to hijacking and spoofing of addresses, etc. The Internet's name resolution infrastructure deeply embeds in its design the assumption of mostly stationary hosts and poorly satisfies the performance, security, and functionality demanded by modern mobile services.

As a step towards addressing these shortcomings, we present the design of a global name service that forms a central component of the MobilityFirst, a clean-slate Internet architecture with *mobility* and *trustworthiness* as principal design goals. MobilityFirst relies on the global name service to cleanly separate identity from network location and to resolve identifiers to locations in a secure manner. More importantly, MobilityFirst capitalizes on the role of the name resolution infrastructure as a logically central, first point of contact to significantly enhance a number of network-layer functions such as supporting host and network mobility, multi-homed traffic engineering, content retrieval, multicast, and next-generation context-aware services. This paper identifies key challenges that must be addressed to realize such a vision and outlines the design of a distributed global name service that can resolve identifiers to dynamic attributes in a fast, consistent, and cost-effective manner at Internet scales.

I. Introduction

The Internet's tremendous success and our maturing realization of its shortcomings have attracted significant research attention towards a clean-slate redesign of the Internet's architecture (e.g., NSF FIND [32], GENI[20], FIA[31]). A number of the shortcomings of the current Internet can be traced back to issues related to *naming*, a central component of any distributed system design. In the current Internet, network entities are identified using IP addresses and the Domain Name System (DNS) resolves human-readable end-host names to IP addresses. Although this design has proven to be surprisingly malleable, it suffers from two sets of fundamental problems, both of which are exacerbated by the the exponential growth of mobile devices and applications today.

The first results from the conflation of identity and location within an IP address, a design decision roundly criticized by many [10], [39], [23], [12], [8]. Using an IP address to identify a network interface as well as the network location of that interface complicates *mobility*—when the location changes but not the identity—and *multihoming*—when a single identity simultaneously resides at multiple locations—e.g., being simultaneously connected to a cellular and WiFi access network. With roughly 5 billion mobile devices worldwide today [19] (over a billion of which are IP-capable) compared to barely a billion tethered hosts [11], mobility and multihoming are the norm, not an exception. Conflating identity and location also poses a serious but less widely acknowledged security challenge, namely, verifying that an interface indeed has the identity it claims. Unlike human-readable names that are bound to public keys by trusted certification authorities in order to enable application-level authentication today, IP addresses are harder to certify, especially when they change many times a day. As a result, we largely make do today with application-level security over a network that can be easily rendered unavailable by spoofing or hijacking of IP addresses.

The second results from the architecture of DNS, a critical part of the Internet's infrastructure. The design of DNS in the Internet's early days implicitly assumed tethered hosts or infrequently changing addresses to be the common case, an assumption evident in its heavy reliance on caching and timeout-based invalidations for scalability. An inevitable consequence of this design is that unanticipated updates to DNS resource records are slow; more than 40% domain names have a TTL of a day or more [38]. Even for slow-changing records, DNS lookup times constitute a significant fraction of user-perceived response times, e.g., over 30% of web objects incur a DNS lookup latency of over a second [24], [22]. Deploying more passive local name sever caches can reduce lookup latencies, but this benefit comes at the cost of further increasing update propagation delays or update load in the system. These and other problems with DNS such as poor load balance and responsiveness to changing demand patterns, vulnerability to denial-of-service attacks, etc. have been well documented in prior research [35], [38], [9], [15].

In this paper, we present the design of a global name service that seeks to address the above problems. This global name service is a central component of MobilityFirst, a clean-

978-1-4673-5494-3/13/$31.00 © 2013 IEEE

slate Internet architecture that is centered around *mobility* and *security* as principal design goals. MobilityFirst relies on the global name service to cleanly separate identity and location and to resolve identities to locations in a verifiable manner. Unlike IP addresses, all identities are represented using flat, self-certifying identifiers, a representation that on one hand does not allow encoding of any location information, and on the other, enables authentication at the network layer via a simple, bilateral procedure.

Our position is that a global name service can and should go beyond identity-to-location resolution and aggressively capitalize on its role as a logically central, first point of contact for most network operations. Accordingly, we describe how a logically centralized but physically distributed global name service can significantly enhance and simplifiy a number of network-layer functions such as seamless host, network, or service mobility, multi-homed traffic engineering, content retrieval, multicast, context-aware services, etc.

A critical distributed systems challenge in realizing such a global name service that supports mobility at scale is the design and implementation of a resolution infrastructure that quickly resolves identifiers to their attributes. To appreciate the scale, consider 10 billion identifiers (for mobile devices, services, content identifiers, or entire networks such as vehicular networks) moving across a 100 network addresses per day, i.e., a load of a million/sec for updates to the network location attribute alone. Furthermore, the name service should process lookup queries quickly, requiring queries to be directed to a nearby replica that holds a consistent replica of the corresponding resource records. Finally, the service should balance the aggregate load across all names and geographically distributed locations of the global name service.

Our proposed design for achieving all of the goals above—low latency, low update cost, and load balance—is a resolver placement engine, Auspice, that automates replication and geo-distributed placement of resolvers for identifiers. To ensure low response times, Auspice dynamically spawns or migrates resolver replicas close to pockets of high demand. To limit update cost, Auspice controls the number of replicas based on write rates.To ensure load balance, Auspice's redirects client requests taking into account both network latency and nameserver load into account.

a) Roadmap: The rest of this paper is organized as follows. Section II presents the naming subsystem in the MobilityFirst Internet architecture. Section III presents the design goals and challenges of an automated replica placement system, Auspice, for geo-distributed name resolution. Section IV describes related work and Section V concludes.

II. Naming in MobilityFirst

In this section, we overview the MobilityFirst architecture with a particular emphasis on how a logically centralized global name service helps achieve a number of network-layer functions in a simple, efficient, and secure manner.

A. Identity and addressing

A *name* in MobilityFirst is a *globally unique identifier* (GUID) that can be used to identify a variety of *principals* such as an interface, a device, a service, a human end-user, content, or (recursively) a collection of GUIDs.

Self-certifying identifiers: A GUID is self-certifying, i.e., any principal can authenticate another principal claiming a GUID without the need for third-party certification. A self-certifying GUID is derived simply as a one-way hash of a public key, so a GUID can be authenticated using a simple, bilateral challenge-response procedure that does not require an external certification authority.

The bilateral challenge-response works as follows. Suppose a principal X (say, a router) wants to authenticate another principal Y (say, a destination), i.e., X wants to verify that Y is indeed the rightful owner of the GUID Y. Then, X issues a challenge by sending a random, one-time nonce n to Y. Y responds to the challenge with the tuple $[K^{+}, K^{-}(n)]$, where K^{+} is a public key and $K^{-}(n)$ is the nonce encrypted using the corresponding private key. Upon receiving the response, X first checks that $H(K^{+}) = Y$, where $H(.)$ is a well-known one-way hash function, and then checks that $K^{+}(K^{-}(n)) = n$. If both checks pass, then X has authenticated Y.

Human-readable names: Self-certifying identifiers do not completely obviate certification authorities. In addition to a GUID, it is convenient to assign a principal an optional human-readable name (e.g., "www.amazon.com" or "Tom Sawyer's cell phone") or an inexact intent (e.g., a set of search keywords or other abstract descriptions). In such cases, *name certification services* bind the human-readable name or intent to a public key, and end-users or applications must first obtain such a certificate from a name certification service they trust in order to securely communicate with each other.

Network addresses: A *network address* (NA) is a self-certifying identifier for a *network*, i.e., an autonomous collection of interconnected devices that act as intermediate forwarders of traffic sourced by or destined to GUIDs attached to any device in the collection. An NA most naturally corresponds to an autonomous domain (AS) in today's parlance, but could also be used to identify finer-grained collections such as a subnet or one or more base stations or coarser-grained collections such as an Internet Service Provider. A GUID is said to be attached to an NA if it is directly connected to one or more forwarding devices in the NA.

B. Routing

The tuple [GUID, NA] is a *routable* destination identifier carried in packet headers. Senders query the name service to obtain the NA corresponding to a GUID (much like they query DNS to obtain the current IP address corresponding to a domain name) before sending the first packet to the destination. Senders are also permitted to send a packet addressed just to a GUID, thereby implicitly delegating to the first-hop router the task of querying the name service for the corresponding NA.

End-to-end packet forwarding is accomplished in two steps, first by an interdomain routing protocol and then by an intradomain routing protocol. The interdomain routing protocol is responsible for delivering packets to the destination NA in the packet header (oblivious of the destination GUID). Once the packet reaches the destination NA, routers in NA are responsible for delivering the packet to the GUID in the packet header. As in today's Internet, all NAs engage in a single interdomain routing protocol, but each NA independently chooses its intradomain routing protocol. As GUIDs can not encode any information about network location, the intradomain routing protocol must be capable of routing on flat identifiers, similar in spirit to a switched Ethernet that routes over MAC addresses.

1) Scaling interdomain routing: The interdomain routing protocol enables reachability to NAs much like the current Internet enables reachability to IP prefixes. Thus, the number of forwarding table entries in a core router is commensurate to the total number of NAs. As the number of NAs may grow significantly over time (e.g., home networks, vehicular networks, body area networks, etc.), the interdomain routing protocol is designed to support a small number of levels of hierarchy so as to trade off packet header space against forwarding table size. Our current interdomain routing protocol design supports a two-level hierarchy wherein networks are explicitly designated as *core* or *edge* networks.

A core network router only maintains forwarding entries for other core networks and a small number of their "customer" edge networks. An edge network router maintain forwarding entries only for a small number of their "provider" core networks and edge networks in their vicinity. The name service enables the two-level interdomain routing protocol by resolving a GUID to a two-tuple $[X, T]$ (instead of a single NA), where X is the most downstream core network enroute to GUID and T is the terminal network to which the GUID is attached. An edge network need not be directly connected to a core network, however, it must ensure that at least one core network agrees to maintain forwarding state for it.

2) Network mobility: The explicit designation of networks as core or edge differs from the current Internet's implicit partitioning of autonomous systems into a hierarchy of tiers. AS tiers in the current Internet are not explicitly recognized by the interdomain routing protocol, BGP, (or even acknowledged publicly by the ASes) and are useful only as means of exposition of the economic relationships between ASes. In MobilityFirst however, the explicit hierarchy also helps reduce global routing traffic under *network mobility* wherein a network as a whole moves across locations, e.g., when vehicular or body-area networks physically move and connect to different access networks.

To enable low-overhead network mobility, each edge network T is responsible for maintaining an entry $[T, X]$ in the global name service. Each GUID u attached to T only maintains the entry $[u, T]$ in the name service. Upon a lookup request for u, the name service checks whether the corresponding network T obtains service from a core network X and, if so, returns the tuple $[X, T]$. If T moves within the service area of X, then no updates to the name service are necessary. If T moves from the service area of X to that of another core network Y, then only the mapping $[T, X]$ in the name service needs to be updated to $[T, Y]$. No updates for individual GUIDs attached to T are necessary.

Network mobility also induces routing update traffic in addition to updates to the name service. When T moves within X (or from X to Y), routers in the service area of X (and Y) have to accordingly update their forwarding table entries. However, routers in all other core networks do not incur any routing traffic because of an edge network's mobility. In contrast, in the current Internet's interdomain protocol, the movement of an AS or IP prefix can induce routing updates to most or all other routers in the Internet.

3) Multihoming and multipath routing: A multi-homed GUID is simultaneously attached to more than one core or terminal network. In these cases, the name service by default returns a list of all homes $\{[X_1, T_1], [X_2, T_2], \ldots\}$ to which the GUID is attached. Multi-homed GUIDs can specify expressive policies for engineering incoming traffic, e.g., *prefer WiFi to 3G*; or *use WiFi for delay-tolerant downloads and 3G for delay-sensitive traffic*, and so on. For multi-homed networks, network operators can explicitly create NAs for portions of their network and specify incoming traffic engineering policies in a similar manner. In contrast, the identity-location conflation in the current Internet forces operators to resort to tactics such as abusing longest-prefix matching in order to accomplish traffic engineering objectives.

The name service also enables multipath routing for multi-homed end-hosts. If a source and destination are homed respectively with k and m homes, then a total of km different routes are potentially available for communication between the source and destination. End-host GUIDs typically specify a default or preferred home in their resource records in the name service for initiating communication. Subsequently, the source and destination can negotiate and use one or more of the other routes as desired.

Compared to the current Internet, the name service in MobilityFirst enables a limited form of user-controlled routing. However, end-hosts can leverage path diversity via the name service only when one of the two endpoints is multi-homed. The conscious decision to not allow full-fledged source routing is motivated by two factors. First, source routing raises security concerns as it is vulnerable to abuse by end-hosts. It is also unclear what incentive a network has to enable end-hosts that are not its direct subscribers to control routes (and by consequence resource allocation) within its network. Second, as in the current Internet, we expect much of the useful path diversity to result from differences in path quality close to the edge (e.g., WiFi vs. cellular vs. wired providers) because of multihoming.

4) Content retrieval: Static content in MobilityFirst is also named using a self-certifying identifier, but a content GUID is simply the hash of the content itself. This widely used technique [13] obviates the need for public keys for verifying the authenticity of static content. Given a content GUID, the name service returns a list of network addresses $\{[X_1, T_1], [X_2, T_2], \ldots\}$ from where replicas of a content may be fetched. The list of these locations typically only includes replicas maintained by content providers or their delegates (e.g., content distribution networks), not all locations storing a cached copy of the content. The client may also request the name service to only return the replica location(s) closest to it, in which case only one or two locations may be returned.

Opportunistic caching and retrieval of static content is enabled by storage-aware routers without explicitly relying on the name service. A network intermediary can intercept a content GUID request and serve the content if it possesses a cached copy. A service identifier field in packet headers allows routers to infer that a GUID is a content identifier as opposed to a self-certifying identifier for other principals. Opportunistic caching alone does not enable routers to leverage wayside copies that are nearby but not directly on the path of a content request packet. We are currently investigating techniques to support efficient discovery of nearby but off-path copies of cached content in intradomain routing protocols.

5) Indirection and grouping: Two powerful operations—indirection and grouping—enable the name service to support a number of new network primitives. Indirection enables the name service to resolve a GUID to another GUID and grouping allow a set of GUIDs to be a assigned a single GUID. We illustrate these benefits using three examples below.

Multicast: A multicast GUID (MGUID) has the same format as a regular GUID and the resolved output of the name service has the same format as multi-homed network address. However, the resolution procedure and routing are different as follows. The name service maintains a membership set for each MGUID that consists of all GUIDs subscribed to the multicast group. Each member GUID i in MGUID subscribes to the group via a single home, NA_i. The name service resolves an MGUID by returning the union of all NA_i's that have at least one GUID subscribed to the MGUID. By default, the sender is responsible for sending data addressed to $[MGUID, NA_i]$ for each of the returned NA_i's. When packets arrive at a destination NA_i, the NA_i is responsible for resolving the MGUID to the subset of member GUIDs attached to its network and forwarding a copy to each member.

Geo-casting, e.g., sending a message to all taxis near Times Square, and other context-aware services can be supported in a similar manner by having the name service maintain geolocation or context attributes in addition to network locations of GUIDs. A number of other context-aware multicast scenarios can be implemented in a similar manner by creating an MGUID for GUIDs satisfying the desired context attribute and resolving the MGUID at transmission time to the constituent destination networks.

Content directories: Content is typically organized in hierarchical name spaces, e.g., www.nytimes.com/sports, www.nytimes.com/business, and so on, to enable grouping and colocation of related content. However, as GUIDs do not capture locality, moving the location of a large content directory from one network domain to another will by default result in updates to all of the constituent content GUIDs. To reduce this overhead, a set of content GUIDs can be assigned a content directory GUID. In this case, the name service maintains network addresses only for the directory GUID and returns it upon a request for any constituent content GUID.

Group mobility: Indirection and grouping help reduce the overhead of updating the name service when nodes move as a group. Note that the name service needs to be updated with network locations of nodes even if they are not actually sending or receiving any data. *Affinity groups*, similar in spirit to content directories, help reduce the overhead of maintaining network locations for any group of GUIDs with similar mobility patterns, e.g., a group of interfaces in an airplane. Thus, any group of co-mobile GUIDs can be assigned a single group GUID that requires only one update for the entire group every time the network location of the group changes.

Affinity group mobility resembles edge network mobility (Section II-B2) as they both help reduce the overhead of updating the name service when principals move as a group. We have distinguished between a group GUID and an NA because of the difference in their roles: members of a group GUID do not act as intermediate forwarders of traffic unlike members of an NA. However, the distinction blurs in the case of ad hoc networks when a group GUID also acts as a network. In such cases, the same group may be assigned a group GUID as well as an NA; routers maintain forwarding state for the latter but not the former.

C. Access control

The name service stores a flexible set of attributes for each self-certifying identifier such as the network address, geolocation, multihoming preferences, group membership information, etc. Principals can further create other attributes as needed to implement new context-aware network services. As some of these attributes may contain sensitive information, it is critical to support access control mechanisms that enable the principal to specify who can read or modify them.

The name service stores data as a key-value store and a principal's self-certifying identifier (GUID or NA) is the primary row key. The data model is a supercolumn family similar to that used by so-called noSQL stores [5]. Each row can have a variable number of attributes, e.g., "type", "network-addresses", "geolocation", etc. For each row-column pair and type of operation (read or write), the name service can maintain an access control policy. The access control policy is specified in the form of a blacklist or whitelist of GUIDs that are allowed to perform the corresponding operation on that attribute. As GUIDs are hashes of public keys, the name service can easily authenticate a principal to verify that it conforms to the blacklist or whitelist. By default, the principal with the primary row key as its GUID has read and write access to all of its attributes.

III. Auspice: A global name resolution system

A key distributed systems challenge in realizing a global name service that achieves the goals described above is the design and implementation of a scalable resolution infrastructure to rapidly resolve identifiers to attributes. In this section, we describe the design requirements and the high-level design of Auspice, a system that automates geo-distributed placement of resolver replicas in a locality and load-aware manner.

A. Overview

MobilityFirst's heavy reliance on the name service to assist network-layer functionality is practical only if querying the name service itself is not perceived as a big overhead. We envision a massively distributed name resolution service that enables any node—an end-host or a router—to obtain a response in a fast and consistent manner. To achieve this goal, it is clearly necessary to deploy name servers at a large number of geo-distributed locations. However, the challenge is to ensure that most requests find a consistent replica of the resource record they seek at a nearby name server location despite frequent updates to the record because of mobility.

We argue that a DNS-like design is poorly suited to addressing this challenge for several reasons. DNS heavily relies on long TTLs both to reduce client-perceived latency and load on the infrastructure, so reducing TTLs will hurt both. However, frequent node mobility implies that TTLs ought to be set to near-zero values in order to ensure consistent responses. Worse, in practice, operators today often ignore TTLs and cache responses for longer than the specified limit, which further exacerbates update propagation times. Finally, sustaining low TTLs for frequent mobility requires authoritative name servers to be sufficiently provisioned and geo-replicated in order to keep lookup latencies low, thereby increasing the cost of maintaining them for end-users and online services.

Auspice automates geo-replicated placement of name resolvers and supports it as an infrastructure service. A *resolver* in Auspice is associated with a single identifier (a GUID) and is responsible for maintaining the identifier's attributes and performing user-request read or write operations. Resolvers for different identifiers are largely independent of each other. For example, the resolver for a popular identifier may be replicated at a hundred different locations while those for an infrequently queried identifier may be replicated at a much smaller number of locations. Different identifiers may also have different consistency, availability, or fault-tolerance requirements. Our position is that supporting automated resolver replica placement as an infrastructure service obviates manual and redundant effort on part of authoritative name service providers, enables them to benefit from economies of scale, and allows the infrastructure service to manage its global resources in an efficient and agile manner.

B. Design goals and challenges

An automated resolver replica placement system must satisfy the following design goals.

1) *Low response time*: Replicas of each resolver should be placed close to its end-users so as to minimize user-perceived response times.
2) *Low update cost*: The number of replicas of each resolver should be controlled so as to limit the update cost required to maintain replica consistency.
3) *Load balance*: The placement of resolver replicas and redirection of client requests should ensure that no single name server location becomes a hotspot.
4) *Fault-tolerance*: A sufficient number of active or dormant replicas of each resolver must be maintained so as to satisfy its availability objective.
5) *Consistency*: The system must achieve the above objectives while respecting the consistency requirements of each resolver.

Although each of the above goals is straightforward and shared by a number of other distributed systems, satisfying the combination of goals is challenging. To appreciate why, consider a few strawman alternatives:

(1) *Replicate everything everywhere*: This scheme can minimize response times but can induce a prohibitively high update cost as well as load imbalance.

(2) *Primary replica(s) plus edge caching*: This scheme maintains one (or a small number of) primary resolver replica(s) and a large number secondary replicas to which updates are pushed out infrequently. This approach can reduce update bandwidth cost by reusing stale, cached copies of resource records, however consistency requirements may prevent or severely limit these cost savings. Even for services with weak consistency requirements, the placement system needs to balance the trade-off between response times and load balance.

(2) *Consistent hashing with replication*: This approach (e.g., [25]) can ensure load balance and fault-tolerance but may incur high response times as the load balance benefits of randomization are fundamentally at odds with placing replicas in a locality-aware manner [38], [14], [37]. Significantly increasing the number of replicas can improve response times but also increase update cost under high mobility.

In accordance with our design goals, Auspice explicitly determines the number and placement of resolver replicas for an identifier taking into account its query and update rates and the geographic locality of queries, and redirects client requests taking into account the aggregate load at each name server. We describe this design in detail below.

C. Auspice design

Figure 1 illustrates the architecture of Auspice. Each identifier is associated with a fixed number, k, 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 k well-known consistent hash functions each of which maps the identifier to a name server location. The replica-controllers form the "control plane" and are responsible only for determining the number and locations of active replicas, and the active replicas are responsible for maintaining resource records and responding to requests from end-users.

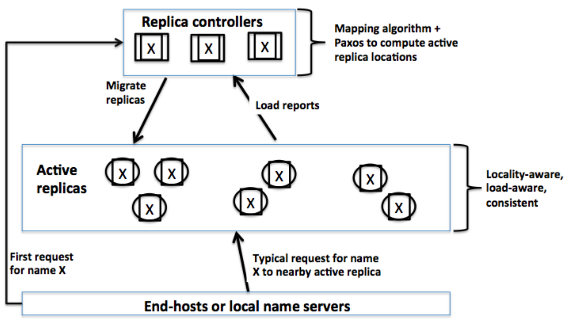
Replica controllers
Mapping algorithm + Paxos to compute active replica locations
Migrate replicas
Load reports
Active replicas
Locality-aware, load-aware, consistent
First request for name X
Typical request for name X to nearby active replica
End-hosts or local name servers

Fig. 1. Overview of Auspice. Clients send typical requests to nearby active replicas of resolvers. Replica-controllers compute the number and locations of active replicas for each name based on load reports in each epoch.

The computation of the active replica locations for each identifier proceeds in epochs as follows. At bootstrap time, the active replicas are chosen to be physically at the same locations as the corresponding replica-controllers. In each epoch, the replica-controllers obtain from each active replica a summarized load report that contains the request rates for that identifier from different *regions* as seen by that replica. Regions could either be terminal networks or geographic regions that partition users into non-overlapping groups so as to capture locality. Thus, each 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 identifier

In each epoch, the replica-controllers use a *mapping algorithm* that takes as input the aggregated load reports and capacity constrains at name server locations to determine the number and locations of active replicas for each identifier. The replica-controllers execute Paxos to compute the placement decision in a coordinated manner for each identifier. During periods of graceful execution, only one replica-controller (the Paxos coordinator) actually invokes the mapping algorithm while the others simply accept its proposed placement; consensus ensures that the replica-controllers maintain a consistent view of the current set of active replicas despite failures.

1) Mapping algorithm: Our preliminary analysis suggests that the mapping problem can be formulated as a mixed-integer optimization problem that is computationally hard. We have also developed simple heuristic placement algorithms that are computationally efficient. A heuristic algorithm that appears promising in our ongoing evaluation is one that creates a number of active replicas proportional to the ratio of the read rate and write rate for the identifier; places some replicas at the locations that receive the highest number of requests for that identifier and places the rest at random locations; and redirects client requests to active replicas taking both round-trip network latency and name server load into account. Our preliminary evaluation suggests that this simple locality- and load-aware replication scheme significantly outperforms simplistic approaches such as "random-k" or DHT-based approaches that do not take locality into account [37]. A detailed description of these schemes and an experimental evaluation comparing these schemes as well as closely related state-of-the-art systems is out of the scope of this architectural overview paper and will appear in a future paper.

2) Routing client requests: The list of all name server locations (i.e., the corresponding [GUID, NA] tuples) is well known and can be obtained by contacting any name server. End-hosts can either directly send requests to the name service or channel them through a local name server like today. When a client encounters a request for a GUID for the first time, it uses the well known set of all name servers and hash functions to determine the replica-controllers for that GUID and sends the request to the closest replica-controller. The replica-controller then returns the set of active replicas for the GUID and the client resends the request to the closest active replica. In practice, we expect replica-controllers to be contacted infrequently as clients can cache and reuse the set of active replicas for subsequent queries.

Clients can also cache and reuse responses if they contain a nonzero TTL, however frequent mobility or consistency requirements may limit the opportunities for such caching. For services whose locations do not change frequently, longer TTLs can reduce name server load and lookup latency, but correspondingly increase service outage time when the service does move. But the explicit separation of identity and location in MobilityFirst helps alleviate this problem. When a GUID disconnects from an NA, it either directly or through the name service (if it disconnects ungracefully) notifies the NA that its routers should remove forwarding table entries for the GUID. If a router in NA subsequently receives a packet destined to [GUID, NA], it responds to the sender with a "refresh resource record" message prompting the sender to query the name service again.

3) Consistency: The name service by default ensures sequential consistency for each resolver service by establishing a total order across all writes to attributes keyed by a single GUID. This is achieved efficiently through Paxos (unrelated to Paxos between primaries for placement decisions as in Section III-C) between active replicas upon a write. When an active replica receives a write to any GUID attribute, it forwards the write to an active replica designated as the current Paxos coordinator. The coordinator selects a sequence number and sends an accept request to all replicas and, upon receiving a successful acknowledgment from a majority of replicas, sends a commit notification to all replicas. Thus, a typical write request incurs four network delays (or two round-trips) to get committed after arriving at an active replica. A read request is processed locally at a replica sees the result of the most recent committed write at that replica.

A total ordering of writes at all replicas is insufficient to ensure a desirable client-perceived consistency property in single-writer scenarios, namely that replicas will eventually return the most current (in real time) network address(es) of a mobile device (the only writer) if no further updates take place, e.g., a mobile may issue update w_2 after update w_1

but it is theoretically possible for the system to commit w_1 after w_2. Ensuring the above property in single-write scenarios requires clients to either issue updates sequentially; or issue multiple outstanding writes through the same active replica; or be responsible for reissuing writes if multiple outstanding writes issued through different replicas get committed in an unacceptable order.

The name service as described above does not guarantee atomicity, isolation, or sequential ordering for operations spanning multiple GUID keys as each resolver is responsible for a single GUID and there is no coordination between resolvers for different GUIDs. For example, in operations involving addition/deletion of a GUID X to/from a group GUID Y, the system may briefly see X as being redirected to Y but Y's membership set not including X or vice-versa. The system may also permanently be in such an inconsistent state if multiple writers concurrently perform multi-GUID operations. Not supporting multi-key transactions is a common design choice in the interest of availability, performance and simplicity made by distributed key-value stores.

4) Mid-session mobility: The design described so far has focused on using geo-replication to enable low lookup latencies at connection initiation time. However, ensuring graceful mobility during a connection's lifetime requires further support for notifying the corresponding endpoints or routers. A well-known approach today to handle mobility both at connection initiation time as well as mid-session is to rely on a home agent [36]; this approach is elegant in that the correspondent remains oblivious to other end-point's mobility, however this elegance comes at the cost of routing inefficiency and the cost of tunneling all data traffic through the home agent.

Auspice places a greater onus on end-points to handle mid-session mobility and only provides a simple mechanism for a mobile node to push invalidating updates to the other endpoint of an ongoing connection when its current address becomes invalid. The mechanism works as follows. Consider a connection between two endpoints with GUIDs A and B where the corresponding sockets are currently bound to addresses NA_A^1 and NA_B^1. If B wishes to migrate its end of the connection to NA_B^2 because of mobility or other reasons, it issues a corresponding invalidating update through the name service to A, prompting A to rebind its socket accordingly. As a common case optimization, it suffices for B to directly issue the invalidating update 'in-band" to A, however the push mechanism via the name service is required to handle the corner case when both endpoints happen to move simultaneously.

The main justification for the above design is that it can achieve low enough connection outage times comparable to a home agent approach when nodes suddenly move mid-session, but without incurring the overhead of triangle routing or tunneling data traffic. We also note that the network-layer support for "refresh resource record" in §III-C2 may also suffice as a notification to the sender, but the above proactive "in-band" scheme above is likely to be more responsive and can also be used when an endpoint wishes to migrate a connection for reasons other than mobility, e.g., from the cellular to WiFi interface. These arguments for using in-band communication for non-simultaneous mobility of endpoints are similar in spirit to several past proposals for session-level [23], [16] or connection-level [7], [30], [43], [18] migration.

IV. Related Work

In this paper, we presented a global name service as a central component of a mobility-centric, secure Internetwork. Our work builds upon an enormous body of prior research that has studied naming, server selection, and placement issues in large-scale distributed systems, as discussed below.

Naming: Classic works on name services [40], [33], [28] for distributed resource discovery have influenced a number of more recent works such as the Intentional Naming System[1] and Active Names[44]. INS[1] proposes a naming system that is also in large part motivated by mobility but has several orthogonal goals. INS allows clients to specify their intention as opposed to a unique name in a simple, high-level language, and the resolution overlay network is tightly integrated with routing, thereby enabling late-binding of intentions to matching destinations. Unlike INS that emphasizes expressiveness of names but not security, our focus is on resolving unique but verifiable identifiers. Furthermore, replication and locality-aware placement of resolvers or wide-area environments are not targeted by INS. Active Names allows applications to deploy mobile code at resolvers that can recursively invoke other programs for finer-grained, hierarchical resolution and compose the output of resolved services. In contrast, our proposed approach does not rely on mobile code and does not attempt to support service composition, but does supports the resolution of identifiers to other identifiers (e.g., for multicast) before eventual resolution to terminal network locations.

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 [17] maps users based on IP addresses to the best server based on latency and server load. DONAR [46] 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 [3] 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: Existing systems that dynamically make replica placement decisions based on observed demand patterns deal with either *static* or *dynamic* content. In general, the latter present a more complex problem than the former as static content can be cached easily obviating sophisticated content placement strategies [26] (noting that static content placement decisions can have a nontrivial impact on network load or cost concerns [6], [41], [42]). Replicating dynamic content is effectively like replicating a service (e.g., the resolver service described in this paper) and must address the interrelated concerns of update cost, consistency, and load balance.

Volley [2] 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. Our preliminary experiments suggest that creating multiple replicas of objects or resolvers can significantly reduce user-perceived response times while also enhancing opportunities to balance load provided update cost is taken into account.

DNS enhancements: Several prior works have studied issues related to performance, scalability, load balancing, or denial-of-service vulnerabilities in DNS's resolution infrastructure [35], [38], [9], [15]. Several DHT-based alternatives have been put forward [38], [14], [37], [34] and we compare against a representative proposal, Codons [38], in our experimental evaluation (not included in this paper). In general, DHT-based designs are ideal for balancing load across servers, but are less well-suited to scenarios with a large number of service replicas that have to coordinate upon updates, and are at odds with scenarios requiring locality-aware placement of replicas. In contrast, our work targets scenarios with potentially high update rates (because of mobility) and seeks to place replicas of resolvers in a locality-aware manner.

XIA: Like MobilityFirst, the XIA future Internet architecture [21] also uses self-certifying identifiers and network locations. Self-certifying identifiers are not new and have been used in a variety of distributed systems [29], [13] as well as specifically in the context of Internet architecture [4], [27]. In both XIA and MobilityFirst, a self-certifying identifier can be used to represent a variety of principals. For evolvability, XIA represents addresses as a directed acyclic graph of self-certifying identifiers. Paths in this graph correspond to possible "source routes" to reach the destination, but not all routers need be capable of processing every identifier and could instead fallback to the default address of the form [GUID, NA]. In comparison, MobilityFirst is designed with more explicit support for mobility-centric services while keeping addresses and packet headers simple. Evolvability comes from the support for indirection and grouping in the global name service that can resolve a GUID to a set of one or more GUIDs.

Vu et al.[45] describe a network-layer-DHT approach to map each GUID to a fixed number of resolver locations using different consistent hash functions wherein clients choose the closest mapped node to resolve requests. This approach is similar in spirit to replacing replica-controllers with active replicas in Section III-C. Our preliminary findings suggest that selecting the number of replicas based on update rates and their placement based on demand locality can significantly improve the performance while keeping update costs low compared to a "random-K" approach.

V. Summary

In this paper, we presented the design of a global name service as a central component of MobilityFirst, a mobility-centric and trustworthy Internet architecture. The name service enhances both mobility and security using self-certifying identifiers that on one hand cleanly separate identity from network location and on the other can be authenticated by any network entity without relying on a third-party. A key challenge we address is the design of the distributed service to resolve identifiers to a variety of attributes in a scalable, consistent, and secure manner. To this end, we presented the design of Auspice, a resolver replica placement system that optimizes user-perceived latency by placing replicas of resolvers close to regions of high demand while respecting capacity and consistency constraints.

References

[1] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., and Lilley, J. The design and implementation of an intentional naming system. In *ACM SOSP* (1999), pp. 186–201.

[2] Agarwal, S., Dunagan, J., Jain, N., Saroiu, S., Wolman, A., and Bhogan, H. Volley: automated data placement for geo-distributed cloud services. In *USENIX NSDI* (2010).

[3] Amazon. Elastic Load Balancing, 2012. http://aws.amazon.com/elasticloadbalancing/.

[4] Andersen, D. G., Balakrishnan, H., Feamster, N., Koponen, T., Moon, D., and Shenker, S. Accountable internet protocol (aip). In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (New York, NY, USA, 2008), SIGCOMM '08, ACM, pp. 339–350.

[5] Apache. Cassandra, 2012. http://cassandra.apache.org/.

[6] Applegate, D., Archer, A., Gopalakrishnan, V., Lee, S., and Ramakrishnan, K. K. Optimal content placement for a large-scale vod system. In *Proceedings of the 6th International COnference* (New York, NY, USA, 2010), Co-NEXT '10, ACM, pp. 4:1–4:12.

[7] Arye, M., Nordstrom, E., Kiefer, R., Rexford, J., and Freedman, M. J. A formally-verified migration protocol for mobile, multi-homed hosts. In *IEEE International Conference on Network Protocols* (2012).

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

[9] Brownlee, N., Claffy, K., and Nemeth, E. Dns measurements at a root server. In *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE* (2001), vol. 3, pp. 1672 –1676 vol.3.

[10] Caesar, M., Condie, T., Kannan, J., Lakshminarayanan, K., and Stoica, I. Rofl: routing on flat labels. *SIGCOMM Comput. Commun. Rev. 36*, 4 (Aug. 2006), 363–374.

[11] CIA. The World Factbook, 2011. https://www.cia.gov/library/publications/the-world-factbook/rankorder/2184rank.html.

[12] Clark, D., Braden, R., Falk, A., and Pingali, V. Fara: Reorganizing the addressing architecture. In *SIGCOMM FDNA Workshop* (August 2003).

[13] Cohen, B. The BitTorrent Protocol Specification, 2008. http://www.bittorrent.org/beps/bep_0003.html.

[14] Cox, R., Muthitacharoen, A., and Morris, R. Serving dns using a peer-to-peer lookup service. In *IPTPS* (2002), pp. 155–165.

[15] DNSSEC. DNS Threats & Weaknesses of the Domain Name System, 2012. http://www.dnssec.net/dns-threats.php.

[16] Ford, A., Raiciu, C., Handley, M., Barre, S., and Iyengar, J. Rfc 6182: Architectural guidelines for multipath tcp development,.

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

[18] Funato, D., Yasuda, K., and Tokuda, H. TCP-R: TCP mobility support for continuous operation. In *IEEE International Conference on Network Protocols* (Oct 1997).

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

[20] GENI. Global Environment for Network Innovations, 2012. http://www.geni.net/.

[21] Han, D., Anand, A., Dogar, F., Li, B., Lim, H., Machado, M., Mukundan, A., Wu, W., Akella, A., Andersen, D. G., Byers, J. W., Seshan, S., and Steenkiste, P. Xia: efficient support for evolvable internetworking. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 23–23.

[22] Huitema, C., and Weerahandi., S. Internet measurements: the rising tide and the dns snag. In *ITC Specialist Seminar on Internet Traffic Measurement and Modeling* (2000).

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

[24] Jung, J., Sit, E., Balakrishnan, H., and Morris, R. Dns performance and the effectiveness of caching. *IEEE/ACM Trans. Netw. 10*, 5 (Oct. 2002), 589–603.

[25] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (New York, NY, USA, 1997), STOC '97, ACM, pp. 654–663.

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

[27] Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., and Stoica, I. A data-oriented (and beyond) network architecture. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2007), SIGCOMM '07, ACM, pp. 181–192.

[28] Lampson, B. W. Designing a global name service. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1986), PODC '86, ACM, pp. 1–10.

[29] Mazieres, D. Self-certifying file system. *PhD thesis* (2002).

[30] Nordstrom, E., Shue, D., Gopalan, P., Arye, R. K. M., Ko, S. Y., Rexford, J., and Freedman, M. J. Serval: An end-host stack for service-centric networking. In *USENIX NSDI* (April 2012).

[31] NSF. Future Internet Architecture, 2012. http://www.nets-fia.net/.

[32] NSF. Future Internet Design, 2012. http://www.nets-find.net/.

[33] Oppen, D. C., and Dalal, Y. K. The clearinghouse: a decentralized agent for locating named objects in a distributed environment. *ACM Trans. Inf. Syst. 1*, 3 (July 1983), 230–253.

[34] Pappas, V., Massey, D., Terzis, A., and Zhang, L. A comparative study of the dns design with dht-based alternatives. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings* (april 2006), pp. 1 –13.

[35] Pappas, V., Xu, Z., Lu, S., Massey, D., Terzis, A., and Zhang, L. Impact of configuration errors on dns robustness. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2004), SIGCOMM '04, ACM, pp. 319–330.

[36] Perkins, C. E. Mobile IP. *IEEE Communications Magazine* (May 1997).

[37] Ramasubramanian, V., and Sirer, E. G. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI* (2004), pp. 99–112.

[38] Ramasubramanian, V., and Sirer, E. G. The design and implementation of a next generation name service for the internet. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2004), SIGCOMM '04, ACM, pp. 331–342.

[39] Saltzer, J. On the naming and binding of network destinations, 1993.

[40] Schroeder, M. D., Birrell, A. D., and Needham, R. M. Experience with grapevine: the growth of a distributed system. *ACM Trans. Comput. Syst. 2*, 1 (Feb. 1984), 3–23.

[41] Sharma, A., Mishra, A., Kumar, V., and Venkataramani, A. Beyond mlu: An application-centric comparison of traffic engineering schemes. In *INFOCOM, 2011 Proceedings IEEE* (april 2011), pp. 721 –729.

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

[43] Snoeren, A. C., and Balakrishnan, H. An end-to-end approach to host mobility. In *International Conference on Mobile Computing and Networking (MobiCom)* (Aug 2000).

[44] Vahdat, A., Dahlin, M., Anderson, T., and Aggarwal, A. Active names: flexible location and transport of wide-area resources. In *DARPA Active NEtworks Conference and Exposition, 2002. Proceedings* (2002), pp. 291 –304.

[45] Vu, T., Baid, A., Zhang, Y., Nguyen, T. D., Fukuyama, J., Martin, R. P., and Raychaudhuri, D. Dmap: A shared hosting scheme for dynamic identier to locator mappings in the global internet. In *Proceedings of IEEE ICDCS* (June 2012).

[46] Wendell, P., Jiang, J. W., Freedman, M. J., and Rexford, J. Donar: decentralized server selection for cloud services. In *Proceedings of the ACM SIGCOMM 2010 conference* (New York, NY, USA, 2010), SIGCOMM '10, ACM, pp. 231–242.