Title

FERN: A Unifying Framework For Name Resolution Across Heterogeneous Architectures

Authors

Sevilla, S.
Mahadevan, P.
Garcia-Luna-Aceves, J.J.

Publication Date

2014-09-01

FERN: A Unifying Framework For Name Resolution Across Heterogeneous Architectures

Spencer Sevilla[a,*], Priya Mahadevan[b], J.J. Garcia-Luna-Aceves[a,b]

[a] *University of California at Santa Cruz, Santa Cruz, California 95064*
[b] *Palo Alto Research Center, Palo Alto, California 94304*

Abstract

A key problem in all name resolution protocols today is that no one protocol performs well across all network architectures. In addition, DNS, the most widespread solution today, depends on a static and connected network layer and faces significant challenges in dynamic wireless networks. We introduce FERN (Federated Extensible Resolution of Names), the first framework designed to enable efficient name resolution across heterogeneous systems operating in dynamic or static networks. FERN organizes nodes into name resolution groups and allows each group to perform name resolution independently and in a manner best suited for that group. FERN arranges these name resolution groups into a hierarchy and allows these groups to communicate efficiently, discover each other's presence, and resolve each other's names. We demonstrate the flexibility and interoperability of FERN by deploying and evaluating it across heterogeneous environments, including a MANET, an infrastructure-based wireless network, and the Internet. We show that FERN performs at least as well as DNS, and yet extends name resolution to networks in which DNS is inadequate.

Keywords: Name Resolution, Service Discovery, Heterogeneous Networks

---

1. Introduction

Service discovery and name resolution are vital operations in any network. Users and applications use text-strings (e.g. URLs), rather than network addresses, to indicate the content or services they require, and these names must then be mapped to network addresses before communication is possible. This requirement applies to today's and future networks and the Internet at large.

Unfortunately, current approaches to name resolution are unable to support this future networking environment because of two constraints. First, no one protocol has been devised that works well across all types of networks. Second, these protocols have not been designed to interoperate with one another. For example, consider the case in which a user accidentally leaves her laptop at home and wishes to access it from the office. The laptop most likely uses multicast DNS (mDNS) [1] to name itself on the home network, but the user has no way of resolving this name outside of that environment, and cannot discover the laptop. As another example, nodes in a MANET may use a distributed protocol to resolve each other's names, but there is no protocol for them to extend this resolution to the Internet through the domain name system (DNS), despite the presence of a network-layer gateway bridging the MANET to the Internet.

To address the above limitations of existing name resolution approaches, we introduce the Federated Extensible Resolution of Names (FERN). To our knowledge, FERN is the first system that provides a low-overhead unifying framework for different name-discovery protocols to interoperate. FERN accomplishes this by organizing nodes into name resolution groups and allowing each group to perform internal name resolution in a manner that works best for that group. FERN also defines a technique for requests to propagate across different groups, and organizes these groups into a hierarchy to support deterministic request-forwarding and ensure reachability across these different name resolution groups. Furthermore, FERN performs these tasks while preserving high scalability and backwards compatibility, and can be deployed incrementally alongside existing protocols such as DNS and mDNS.

Section 2 provides an overview of prior work in name resolution and service discovery for different types of networks. Section 3 specifies the operation of FERN within a group and across groups, and Section 4 examines caching and forwarding in FERN. Section 5 provides formal proofs of FERN correctness and operation. Section 6 examines several different topics related to a real-world FERN deployment, and Section 7 describes our testbed implementation and provides preliminary results collected with this system. Section 8 concludes the paper.

2. Related Work

Prior work in name resolution and service discovery can be loosely divided into client-server systems, peer-to-peer

---

[*] Corresponding author email: spencer@soe.ucsc.edu

systems, or systems based on overlay networks. Additionally, there exists work on hybrid systems employing more than one of these architectures and on interoperability between these systems.

2.1. Client-Server Systems

The most prominent approach to name resolution today is the DNS[2]. DNS relies on a hierarchy of servers that must be configured to forward a name-request to the appropriate server, which then resolves that name-request to an IP address. Through the use of this hierarchy, load-balancing "secondary" servers, and caching, the DNS provides name resolution for the entire Internet today. However, this scalability comes at a price. First, the DNS relies completely on these servers: if the authoritative DNS server for the subdomain `example.com` is down, overloaded, or configured incorrectly, then all DNS lookups for `*.example.com` fail and `www.example.com` is not reachable, regardless of the state of the web server itself. Refer to [3, 4] for recent examples of DNS outages that affected millions of users. Second, given that the DNS relies on hosts to configure their IP address with their DNS server using out-of-band communication, it is a static system that cannot support dynamic networks. Dynamic DNS [5] seeks to alleviate these limitations by specifying an UPDATE record type; however, it still requires that (1) the host knows the IP address of its authoritative DNS server *a priori*, and (2) the host successfully sends an update to the authoritative server every single time its IP address changes.

2.2. Peer-to-Peer Systems

Peer-to-peer systems such as mDNS [1], SSDP [6], and SLP [7] do not require a central server to operate. As a result, these systems require minimal configuration (hence the name "Zeroconf") and are well suited to dynamic environments where hosts come up, go down, and change IP addresses frequently, such as home networks configured with DHCP [8] or AutoIP [9]. Unfortunately, all peer-to-peer systems currently share a heavy reliance on IP multicast to propagate both name-requests and service announcements through the entire network. As a result they suffer from relatively high latency and cannot scale, which restricts these protocols to LANs, where internal names are denoted by the top level domain (TLD) `.local`.

2.3. Overlay Networks

Several publications [10, 11, 12, 13, 14, 15] discuss deploying DNS over an overlay network that uses a distributed hash table (DHT) to reduce the load on individual servers and thus provide higher scalability and better fault tolerance. These papers target the traditional DNS system in the Internet, and thus focus on planet-level scalability. They present mixed results on latency, but note that DHTs serve to decouple the physical location of an entry from its logical location. This architecture helps with load-balancing, removes hot spots and bottlenecks in the hierarchy, and creates a system that is orders of magnitude harder to attack. These benefits are typically achieved by enforcing a flat namespace, where all records in the system are stored as equal objects in one giant DHT. Unfortunately, these approaches rely on a network environment in which the nodes of the overlay are static and available with high uptime, the topology is connected, and links have plenty of bandwidth. The performance of DHTs degrades significantly in dynamic networks as a result of excessive overhead resulting from topology-independent overlay addresses, link failures, and node mobility.

2.4. Hybrid Systems

There are a number of hybrid approaches to name resolution that attempt to combine the architectures described above. SLP, for example, introduced the concept of an optional "Directory Agent" (DA) that nodes in a network must contact first if it is present. Kozat et. al. [16] bring this concept to the case of MANETs by proposing a virtual backbone of "Service Broker Nodes" (SBNs) that form a dominating set in a MANET and proactively maintain routes through the MANET to each other. These proposals attempt to increase scalability by only allowing a select subset of nodes to query the entire network, and requiring that other nodes communicate with their closest directory node. However, these approaches all share the same drawback, which is that communication between directory nodes is unstructured and accomplished by flooding a name request to all other directory nodes, which scales as poorly as Section 2.2. MDHT [17] addresses this issue by proposing a hierarchy of DHTs, but cannot scale to large numbers of records because it requires the top-level DHT to contain every record in the system.

2.5. Interoperability

A unifying problem of every approach described above is that every node must be a member of the protocol for name resolution to occur. This lack of interoperability means that the two protocols cannot talk to each other, even though mDNS might be best for home networks and DNS might be best for the Internet. Currently, support for multiple protocols is accomplished by designating some top level domains or TLDs (such as `.local`) for certain protocols and having the node generating a request use the TLD to decide which protocol should be used. A few approaches [18, 19, 20] have been published on interoperability between multiple resource-discovery protocols, but these works have been limited to developing higher-layer application programming interfaces (APIs) that mask implementation differences between protocols that already share the same basic architecture, such as SSDP and SLP.

Plutarch[21] proposes an architecture for interoperability across different network architectures, both for routing and name resolution. Instead of requiring all networks to use the same protocol, Plutarch divides networks into

Function Prototype	Comments
int (0 = success) joinGroup(args)	args varies as a group-specific parameter
int (0 = success) leaveGroup()	Groups must also support ungraceful departures
int (0 = success) registerName(name)	name is not fully-qualified (i.e. just "printer")
network_address resolveName(name)	name is not fully-qualified (i.e. just "printer")
network_address getParent()	assumes the parent group can be reached at this address:udp53
network_address getChild(name)	same as above, but returns (null) if it has no child with this name
int (0 = success) registerChild(name)	name is not fully-qualified (i.e. just "lab_3")
int (0 = success) deregisterChild(name)	name is not fully-qualified (i.e. just "lab_3")

Table 1: FERN Name Resolution Group API

contexts and proposes the use of interstitial functions to translate between contexts, similar to the way network address translation (NAT) is implemented today. Though Plutarch provides a model for interfacing radically different network architectures, it effectively leaves the implementation of these interstitial functions "to the reader." Plutarch also raises an important question about how the different contexts become aware of each other: it proposes using a gossip protocol to disseminate this information, yet this protocol might cause issues of scalability and coherency if the number of separate contexts becomes too high or if entire contexts exhibit a high degree of mobility.

3. FERN: A Unified Framework For Name Resolution

FERN specifically targets the problem of protocol interoperability described in Section 2.5. Rather than proposing one particular name-resolution protocol or architecture to be used by all networks, FERN proposes a solution where different groups of nodes may use different name-resolution protocols and still communicate.

FERN achieves this solution by providing a framework for interoperability among different name resolution protocols, such as the ones described in the previous section. FERN organizes nodes using a common name resolution scheme into separate *Name Resolution Groups* (NRGs), and specifies a protocol for NRG intercommunication. FERN then organizes the NRGs into a hierarchy. The primary motivation for organizing nodes into NRGs is to: (a) separate nodes that use different name resolution schemes; and (b) reflect the natural groupings that appear in the underlay network (i.e., subnets), logical hierarchy (i.e., org charts), and users themselves (i.e., social groups). FERN defines a set of operations that a group must support, but explicitly does *not* define the implementation of these operations. FERN assumes that all nodes in an NRG are able to exchange messages at the application-layer, but does not assume that all the nodes run a specific network-level protocol.

3.1. NRG Responsibilities

The first responsibility of a FERN NRG is that every node in a group must be able to resolve names for which the group is responsible. Similar to DNS, NRGs are responsible for names that end in the NRG's fully-qualified name. For example, an NRG named `lab_3` is responsible for queries such as `printer.lab_3` or `johns_pc.lab_3`. To facilitate these responsibilities, NRGs must provide a way for its members to: (1) register names, (2) resolve names, (3) join the NRG, and (4) leave the NRG.

The second responsibility of a FERN NRG is that it must forward queries for which it is not responsible. This requirement is accomplished by organizing the NRGs into a naming hierarchy and allowing NRGs themselves to be members of another NRG. In this situation, the *children* of an NRG are its members, and the *parent* of an NRG is the NRG of which it is a member. This relationship is denoted using the same dot-notation as in DNS. For example, an NRG with the name `lab_3.parc.usa` is a member of the NRG `parc.usa`, which is itself a member of `usa`. For clarification and brevity, in the remainder of this paper, the *shortname* of this NRG is `lab_3`, whereas its *fullname* is `lab_3.parc.usa`. This child-parent relationship between NRGs creates a *name resolution tree* (NRT) as in the DNS, with the *root NRG* "/" at the top, and this tree powers the forwarding of requests among NRGs. Though the NRG `lab_3.parc.usa` in our example is not responsible for the query `x.ccrg.ucsc.usa`, it can forward the request up the NRT to `usa`, and then down to `ccrg`. Furthermore, `lab_3` can perform this task without knowing the network address of `ccrg` itself; all NRG `lab_3` needs to know is how to contact its children NRGs and parent NRG. We express FERN programmatically as a pseudocode API in Table 1, and formally define the set of rules in List 1.

3.2. NRG Communication

For the sake of interoperability with DNS, we have chosen to use the traditional DNS record format (A, CNAME, . . .) and port (UDP 53). This choice means that to support request-forwarding along a branch in the NRT, all an NRG has to store is the network address of the other NRG. This results in an exceedingly simple interstitial function, and means that inter-group resolution through the entire hierarchy can be supported by simple recursion.

Figure 1 shows an example of request forwarding in FERN. Here, the NRG `example` uses a server, `subgroup1`

List 1: FERN NRG Rules

1. NRG X has at most one parent NRG Y in the FERN NRT, and $fullname_X = shortname_X.fullname_Y$.
2. NRG X can have several child NRGs in the FERN NRT, and each of these children has the full name $childname.fullname_X$.
3. NRG X must be able to communicate with its parent and children NRGs in the FERN NRT.
4. NRG X must know the addresses of all its ancestor NRGs in the FERN NRT.
5. NRG X is responsible for directly answering all queries that end in $fullname_X$.
6. NRG X must forward queries to the best match of NRG possible, adhering to the caching rules in List 2.
7. NRG X must return an error for a query that it cannot answer or forward.

uses request-flooding, `subgroup2` uses a DHT, and a smartphone in `subgroup1` wants the address of the printer in `subgroup2`. First, the request is flooded through `subgroup1` until it reaches a node that can communicate with `example`. Next, the interstitial function of FERN is used to forward the request up to `example`, and then again to forward the request down to `subgroup2`. Lastly, `subgroup2` uses its DHT to resolve the address of the printer. This behavior is contrasted with Figure 2, which shows the same name resolved iteratively in DNS. As Figure 2 illustrates, DNS requires that (1) each name group be supported by an authoritative name server, (2) resolution starts at the root server and descends the NRT, and (3) servers support iterative resolution, where the resolver communicates with each name-server in turn.

3.3. Internal Group Policies

The FERN architecture places no constraints on the number of services or names an individual node may register, the nature of these services, or the number of NRGs of which a node may be a member simultaneously. It is left to individual NRGs to implement and enforce rules such as restricting group membership to certain nodes or restricting the names that a particular node may register. NRGs may choose to adopt and enforce certain naming conventions (similar to the mDNS service registry), and these conventions may even standardize across different NRGs; however, this process is outside the scope of FERN.

FERN treats group security the same way. NRGs may choose to use encryption, MAC addresses, or other out-of-band information to authenticate, authorize, and verify their members and names. They may also decide to use name resolution to enforce other security policies, such as only allowing certain nodes to resolve the address of certain services. However, the administration and implementation of these policies are left to the individual NRG, not the entire framework.

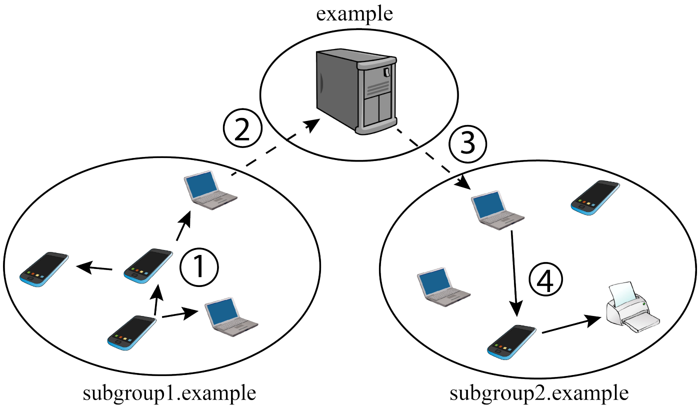

Figure 1: Request-Forwarding Across Groups

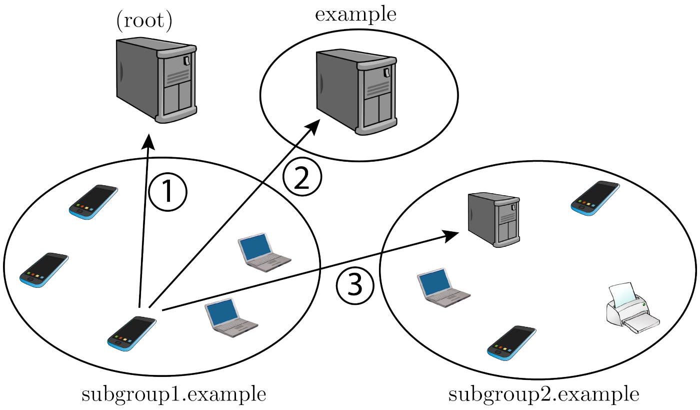

Figure 2: Request-Forwarding In DNS

3.4. Group Size

FERN also places no explicit limits on the size of an NRG; this applies to both the number of nodes in an NRG and the physical distance between nodes. However, all NRGs must conform to the rules outlined in Section 3.1 and List 1. These rules may create an implicit, functional limit on group size, but this limit depends heavily on the particular architecture and implementation powering the NRG.

3.5. Bootstrapping Group Membership

For a node to join an NRG with the `joinGroup(args)` operation in Table 1, it must already know (1) the group architecture, (2) any `args` the group requires, and (3) to whom to send this information. Though the mechanics and specifics of joining an NRG should be handled by the NRG itself, the process of group discovery and acquiring the information listed above must be standardized, because it is a process that exists outside of any individual NRG and may interact with other protocols and systems. There are several protocols (e.g. DHCP [8] and AutoIP [9]) used to help nodes join a network by supporting discovery, authentication, and address acquisition. They also

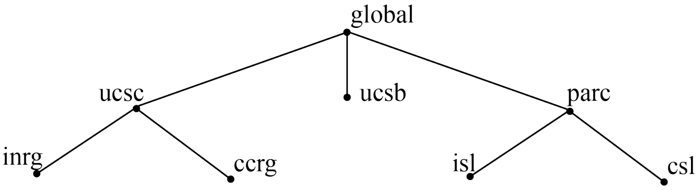

Figure 3: Example FERN Name Resolution Tree

bootstrap DNS resolution by providing hosts with the address of a local DNS server to be used. Thus, FERN can extend these existing protocols by defining an extra FERN record to be passed to a node when it joins the network. This record should contain: (1) the fullname of the NRG, (2) the structure of the NRG, (3) any group-specific arguments, and (4) a fallback network address to be used as a local DNS server if the node does not recognize the value in (2) or is FERN-unaware.

3.6. NRT Traversal

The FERN process of forwarding requests up and then down the NRT also affects the fault tolerance and resilience of the system. In DNS, if a node is unable to contact the root server it is unable to perform any resolution, as shown in Figure 2. This behavior makes the root server an attractive target for attackers, and also restricts the usefulness of DNS to nodes that can access a root name server, as opposed to nodes in a private network or MANET. Conversely, FERN requests only travel up the NRT as far as necessary. Thus, in the example NRT shown in Figure 3, the only queries that would reach the root NRG are requests from NRG `ucsc` to NRG `parc` or vice-versa. All other traffic stays within either NRG, and thus would function normally independently of their ability to access the root NRG.

3.7. Internal Resolution

By forwarding queries in the manner described above, FERN reduces the reliance on the top-level NRGs of the NRT and improves resilience among lower-level NRGs. If a root or TLD server fails, or if a NRG is cut off from these servers due to a network partition, internal resolution is unaffected. When combined with FERN's support for different internal name-resolution architectures, this forwarding behavior enables FERN to effectively power name-resolution in dramatically different network-layer architectures. Such architectures could include opportunistic, delay-tolerant, or pocket-switched networks, depending on the name-resolution architecture.

By supporting incredible diversity in name-resolution protocols, FERN enables a corresponding diversity in network-layer protocols. Ideally in FERN, if there exists a network-layer route between two hosts, they should be able to resolve each other's names and communicate. Conversely, if no route exists between the hosts, then name resolution is unimportant because even in the event of successful resolution, no communication can occur.

3.8. Caching

Caching name responses and intermediate name referrals significantly reduces latency and overall network load. FERN enables caching by allowing groups to append a resource record for their group itself whenever they answer a query or recursively return the answer to a query. Thus, if a request originates at group A and traverses groups B, C, and D before finishing at E, the requesting node could end up caching the network address of groups B through E if these groups elect to append their network address to the response. Additionally, intermediate groups may also read these records, so in this example group C could also learn the network address of groups D and E.

As in DNS, intermediate record caching can significantly change system performance and behavior. Since FERN differs from DNS in the way it forwards queries and responses, these changes in system behavior have the potential to be much different than in DNS. A comprehensive analysis of these differences and necessary policies is provided in Section 4.3.

3.9. Fault Tolerance And Resilience

Consider the case where the NRG `parc` fails. In DNS, all nodes (including nodes within the domain `parc`) would be unable to resolve any names below domain `parc` in the tree, but are able to resolve all other names. Conversely, in FERN, requests that stay inside NRGs `csl` or `isl` would still succeed, but none of the nodes in these NRGs would be able to resolve any names outside of `parc`, unless the NRT is modified to reflect the failure that took place.

FERN addresses this problem by allowing nodes to cache the network address of other nodes in their ancestor NRGs all the way up to the root of the NRT. An NRG may use these addresses to forward requests to its grandparent if and only if its parent is unresponsive. With this rule in place, FERN may often do better than DNS (by preserving internal resolution when possible) but it never does worse, since it effectively reduces to DNS when intermediate NRGs fail.

3.10. Internal Redundancy

To mitigate the risk of a node failure, NRGs may choose to replicate records across $K > 1$ separate nodes. In this case, choosing a proper value for K depends heavily on the underlay network powering the NRG. In the case of the Internet, the DNS itself shows that small values of K are sufficient. For example, over 80% of DNS entries were supported by just one or two name servers in 2004 [22]. In other, more-dynamic networks such as MANETs, higher values of K might be necessary to ensure continued resolution and operation in the face of node failures or network partitions.

Another important factor for replication is the position of the NRG in the NRT. As implied in Section 3.9, NRG failures higher in the NRT can be more problematic and disruptive to end-to-end resolution because a failed NRG

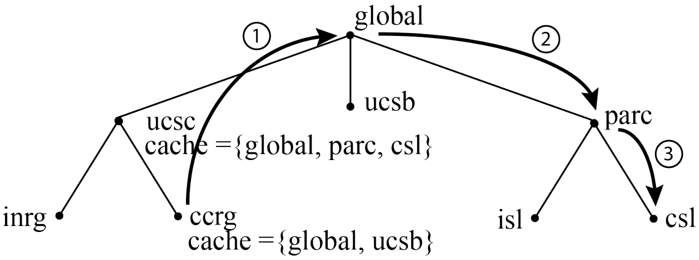

Figure 4: Ubiquitous Caching Policy

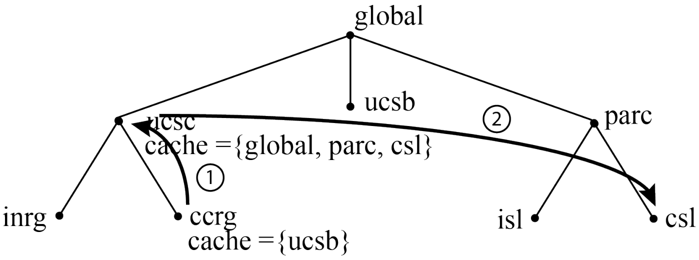

Figure 5: Caching Behavior With Rules

cannot forward queries. In these cases, it is important to ensure that K is large enough that the NRG maintains sufficient uptime requirements. However, in other network scenarios such as bridging a MANET to the Internet, $K = 1$ might be completely acceptable if the only node bridging name-requests is also the only node able to perform network address translation: in this case, higher values of K accomplish nothing because if the bridging node fails, the underlay network itself is partitioned. In this light, ideally K should be sufficiently large so that name resolution reflects network connectivity, and is never the bottleneck or weakest link.

4. Forwarding and Caching Policies

The primary difference between FERN and DNS is FERN's ability to support different group architectures, which is explained in detail in Section 3. However, more nuanced differences between the systems include (a) how the system forwards and handles recursive queries and (b) how this difference affects caching policy.

4.1. Recursion Vs Iteration

Figures 1 and 2 illustrate that the DNS generally employs *iterative* resolution, whereas FERN enforces *recursive* resolution. Though the DNS does provide a recursive resolution option, it is typically disabled, and explicitly must be for higher-level domains such as TLDs. This design decision is primarily motivated by security concerns, since a malicious query can exploit a recursive DNS server, either targeting the server itself or using the server to launch an amplification attack on another target.

4.2. Forwarding Queries

In addition to mandating recursive resolution as opposed to iterative, FERN also differs from DNS in the path that queries take through the NRT. Even when employing recursion, DNS queries always start at the top of the tree and descend down, as opposed to FERN queries which first climb the NRT until a common ancestor is reached. This fundamental difference in forwarding behavior is a vital part of FERN, and enables operation in disrupted or non-Internet-enabled networks such as MANETs, since it removes the DNS dependence on TLD and root servers. In these networks, the only comparison that can be made is that FERN succeeds where DNS fundamentally cannot. In contrast, the difference in forwarding policies is much more apparent in the case where an NRG is a part of the Internet. In this case, a node could resolve a hostname using either FERN or DNS, and the *only* real difference between the two protocols is that FERN climbs the tree to a common ancestor.

4.3. Caching Policy

The difference in forwarding policies discussed above affects which groups in FERN, or servers in DNS, receive intermediate requests and responses, and this in turn affects caching policy. In DNS, resolution always starts at the top of the NRT and descends down the tree. This difference means that a cache-hit always results in reducing traffic on a higher-level server. For example, a cached entry for `ucsc.edu` avoids sending queries to the root server or the `edu` TLD. Thus, caching in DNS only serves to reduce load, especially on higher-level servers, and can never worsen performance.

While this point appears obvious, it stands in stark contrast to FERN, which requires a much more careful caching policy. In FERN, since a request first climbs the NRT before descending, a careless cache-hit may inadvertently result in *more* work being done by the system. This point is best illustrated by Figures 4 and 5, which show an example NRT with caching enabled. In both examples, a node in `ccrg.ucsc` wishes to resolve a name in `csl.parc`. In Figure 4, the group `ccrg` has a cache entry for the root group, `global`, and so the request is forwarded directly to the root group and bypasses the better cache-hit in `ucsc`. In contrast, if the cache entry for `global` is *removed* as in Figure 5, we see better performance in two key ways: first, the total number of hops is reduced from 3 to 2. Second, and more importantly, we avoid having to issue a request to the top group `global`.

The above examples illustrate the need for a more refined caching policy in FERN, since ubiquitous caching as in DNS will create the scenario in Figure 4 very quickly. With ubiquitous caching, once any node in `ccrg` resolves any name outside of `ucsc`, the top group is queried, and its address cached upon successful resolution, as described in Section 3.8. After this caching occurs, any time any node in `ccrg` wishes to resolve any name outside of `ucsc`, the closest matching cache-hit will always be `global`. Moreover, this creates *another* problem: if `ccrg` queries `global` directly as in Figure 4, then the group `ucsc` is not on the return path and has no opportunity to cache the address of `parc` or `csl`. This lack of cached information means that a node in `inrg` wishing to resolve the same name cannot take advantage of any cached information, and the request must ascend and descend the tree as before.

The solution to this problem is to restrict the ability of certain NRGs to cache the address of other NRGs. In Figure 5, if a caching policy prohibits `ccrg` from caching the address of `global`, we can ensure proper operation and avoid the scenario in Figure 4. Furthermore, we ensure that in the event of a request climbing and descending the NRT, more intermediate groups are able to cache responses. With these goals in mind, we designed a set of FERN caching rules, stated in List 2.

These rules significantly reduce the load on nodes in NRGs that are higher in the hierarchy and serve to create a much more distributed system. Revisiting the above example, they ensure that the group `ccrg` will never cache the address of `global`, which means that instead of querying the group directly, it must recursively ask the group `ucsc` to resolve a name in `parc`. Asking the group `ucsc` has the benefit of ensuring that in the event of a successful response, `ucsc` is on the return path and can cache the address of `parc`. Placing `ucsc` on the return path ensures that (a) the name is cached, so the group `global` only has to be queried once and (b) the cache entry in `ucsc` will be accessed by all nodes within `ucsc`, rather than some nodes "leapfrogging" the group. This policy helps distribute caching information throughout the NRT, and helps to reduce traffic on higher level name-servers.

4.4. Hybrid Behavior

Caching leads to behavior that closely resembles a hybrid system. In the examples above, consider a scenario where the bottom groups use architectures better-suited for dynamic networks. The first time a node in one of these groups attempts to resolve a name outside of the group, it must call `getParent` and use the group to internally resolve the address of its parent. However, the resolving node may then cache this address and send all future requests directly to its parent group without needing to re-resolve its address. This behavior is remarkably similar to the hybrid approaches described in Section 2, where local requests stay local and system-wide requests get forwarded to the appropriate SBN or DA, yet FERN enables this behavior without the added protocol complexity of specifying how it should be done, figuring out what constitutes a local request, or forcing that system on all network scenarios. This behavior can also be compared to current systems, where requests are either multicasted over mDNS or sent to a local DNS server based on the TLD of the name-request. FERN exhibits very similar behavior, yet accomplishes this without fragmenting the namespace.

List 2: FERN Caching Rules

1. A node in NRG X may cache the address of nodes of NRGs for which NRG X has a branch in the FERN NRT.
2. A node in NRG X *may not* cache addresses of nodes in NRGs that are closer to the root of the FERN NRT.
3. A node in NRG X may cache the addresses of nodes in NRGs that are at the same level of NRG X in the NRT, or further down the NRT.

4.5. FERN and Plutarch

Given that FERN does not specify the internal mechanics of an NRG, it can be compared to Plutarch, in that individual NRGs can be compared to Plutarch's contexts, and request-forwarding to the interstitial function. However, FERN differs from Plutarch in how naming and name-resolution is performed. Plutarch stresses that names need not be global, yet implicitly requires globally-unique *context* names, since name-resolution starts by specifying a context. Moreover, Plutarch essentially proposes using a gossip protocol and request-flooding to disseminate context information, which leads to questions of scalability as well as deterministic resolution.

In contrast, FERN explicitly addresses the problem of locating a particular named context by organizing groups into a hierarchy. In addition to supporting globally unique names, this hierarchy ensures that: (1) the system resolves names deterministically, (2) name requests do not traverse NRGs unnecessarily, and (3) scalability is preserved by enforcing an upper-bound on the number of other NRGs any one group must know.

5. FERN Proofs Of Correctness

The FERN rules in Lists 1 and 2 allow us to formally prove that requests processed in FERN deterministically terminate, do not loop, and are resolved correctly. The following proof focuses solely on loops resulting from the misconfiguration of FERN groups, and does not consider underlay network errors or malicious behavior.

Theorem 1: FERN request forwarding is loop-free.

Proof: Assume that there exists a request-forwarding loop among nodes using FERN. Given that NRGs are organized as a tree, which is acyclic, the existence of a request-forwarding loop necessarily implies that an NRG i must forward the request to another NRG k that is not its parent or child in the FERN naming tree. However, according to Rule 6 in List 1 and List 2, an NRG that forwards a request must do so to either (a) its parent or (b) one of its children. Given that the NRT is acyclic, the request-forwarding loop must occur as a result of misconfiguring either the parent NRG or one of its children.

According to Rules 1 and 3, an NRG cannot mistake its parent and hence NRG i cannot consider NRG k to be its parent NRG mistakenly. This means that the request-forwarding loop must result from the misconfiguration of a child NRG, i.e., assuming that NRG k is a child NRG when in fact it is not. For this to be the case, NRG i must know how to contact NRG k, and NRG i can acquire this knowledge only through the registration process. Since the registration process is always initiated by the child NRG, it is impossible for NRG i to mistakenly assume that NRG k is its child when it is in fact not the case. This completes the proof.

Though Theorem 1 may appear trivial, referral loops are possible in the DNS, and these loops significantly impact network performance. Jung et. al. [23] observe that a very small portion (3%) of requests to misconfigured nameservers result in referral loops, and these requests generate over 12% of all DNS packets, on average retrying each query over ten times before giving up. Theorem 2 below serves as a general proof of correctness for FERN requests.

Proposition 1: Any two NRGs have a common ancestor.

Proof: The proof is immediate from the fact that the root group "/" is the parent of all TLD groups, hence it is a common ancestor of every group.

Theorem 2: FERN name resolution is provably correct

Proof: Without loss of generality, assume that some node in group X wishes to resolve the name of a node in group Y. From Proposition 1, it follows that groups X and Y must have a common ancestor; call that ancestor group Z. By Rule 3 and Theorem 1, the request originating in group X can be forwarded up the tree until it reaches group Z. Again by Rule 3 and Theorem 1, once the request reaches group Z, it is forwarded down the tree until it reaches group Y, which resolves the request. Hence, any node in any group is always able to resolve the name of any node in any other group.

6. Deployment Concerns

The previous sections have focused primarily on defining the underlying behavior of a FERN system. In this section, we address other issues that would affect a large-scale FERN deployment, such as performance, scalability, security, and resilience to attacks.

6.1. Internal Group Communication

The best choice for internal group communication depends on both the underlying network topology and the number of nodes in the NRG. Though an NRG may specify that only a certain number of nodes may join, the number of nodes in an NRG is determined primarily by external factors, which in turn determine group communication. These external factors could be logical (the number of people in an organization), hierarchical (an org chart), or based on the underlying network topology (e.g., nodes in a MANET).

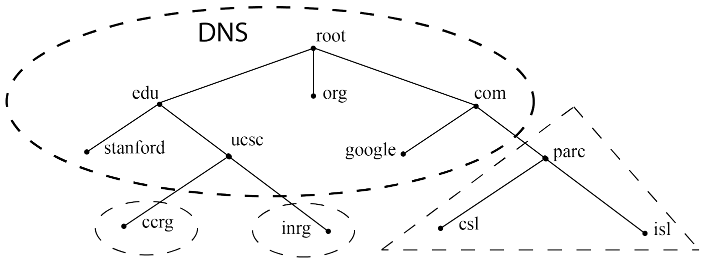

Figure 6: DNS Integration With FERN

In the case of the Internet, a connected underlay network with static addresses, the client-server architecture has been shown by the DNS to be efficient, scalable, and provides an attractive first choice, as discussed in Section 6.2. For fully-connected networks with dynamic network addresses (such as an internal subnet or home network), a DHT may be a better choice for both robustness and dynamic updating. As we continue along the spectrum to more dynamic networks such as MANETs, it is likely that highly specialized protocols will evolve to complement different routing protocols, such as AODV or OLSR.

6.2. DNS and FERN Interoperability

Conceptually, DNS can be thought of as a subset of FERN, since DNS zones are roughly equivalent to FERN's NRGs. Because FERN explicitly does not define the implementation details of an individual NRG, an NRG could easily be powered by DNS servers configured to support a zone of the same name. This interoperability is enhanced by the fact that FERN uses the same record format and port as DNS.

However, if FERN NRGs are to be deployed in a system alongside DNS, the DNS domains and zones *must* be placed at the top of the hierarchy to ensure proper operation. This requirement is because of the differences in caching policies and forwarding between FERN and DNS: if a FERN NRG at the bottom of the hierarchy is actually powered by DNS servers, it will send requests directly to the top level of the hierarchy. This behavior then creates the caching issues described above.

These problems do not occur if all the NRGs powered by DNS form a contiguous set of NRGs that includes the root of the NRT. An example is shown in Figure 6, with the dotted lines indicating separate FERN NRGs. In this case, FERN can effectively treat this set of NRGs as one large system that resolves names and forwards queries appropriately.

This topology is important because it strongly resembles how FERN would interact with the DNS if it were to be deployed today. Rather than seeking to *replace* the DNS, FERN can be thought of as *extending* the DNS, in particular to deal with networks where the DNS architecture is inappropriate. FERN enables a scenario where the core of the NRT is powered by the DNS, and network-specific NRGs are deployed in lower regions of the tree.

6.3. NRT Height

The current DNS hierarchy is relatively shallow, with a typical height of three or four levels, but is almost exclusively limited to naming Internet servers. We believe that a full FERN NRT would have more levels, because part of the intent of FERN is to expand name resolution to devices in different network environments. As described above, the addition of NRGs in the NRT could be the result of several logical or organizational factors, as well as underlay network concerns (such as bridging resolution across two MANETs). In Section 7 below, we investigate the performance overhead of adding extra groups to the NRT.

6.4. Scalability

A primary concern with any name resolution protocol is scalability, and this is one area where the DNS performs exceptionally well. The DNS today is clearly Internet-scale, and yet manages to accomplish this while keeping latency low, typically under 100ms total for a query and response [23]. This performance sets a high bar for the evaluation of new name resolution protocols, which typically propose trading one characteristic for another. For example, DHT-based approaches (described more extensively in Section 2.3) are typically orders of magnitude more robust and resilient to attack, yet accomplish this with untenable increases to average latency.

As mentioned in Section 4.2, no quantitative comparisons can be made between DNS and FERN in the disconnected scenarios where DNS completely fails. Thus, to compare FERN to DNS with regards to scalability, we must assume a large, connected underlay network scenario suitable for DNS. In this scenario, the differences in recursion, forwarding, and caching policies may play a large role in the overall performance of the system; we evaluate these metrics using a small topology in Section 7. However, the interoperability feature described in Section 6.2 effectively makes the DNS a "lower bound" on FERN performance in these network scenarios: if a FERN system struggles to achieve Internet scale, the solution is to simply *use* DNS to power this part of the NRT. Thus, we conclude that while FERN may do better than DNS depending on several factors examined below, it may never do worse.

6.5. Denial of Service Attacks

A primary attack vector for the DNS is an amplified DDoS, which takes advantage of a DNS server that has been configured to allow recursive resolution. The only way to protect against these attacks is to (a) disable recursion or (b) restrict service to a trusted set of clients. Unfortunately, option (b) is infeasible due to the nature of the DNS, since queries may come directly from end-hosts or local DNS servers, both of which are deployed without registering anything in the DNS itself. Somewhat ironically, another obstacle for top DNS servers to only serve trusted clients is that requests may come from anywhere, specifically *because* the DNS must also support iterative resolution.

In contrast, FERN is able to easily implement option (b) specifically because it enforces recursive resolution. Consider, for example, the TLD `.edu`. In FERN, the only entities that may send a query to `edu` are its direct children nodes and NRGs, such as `ucsc.edu`. However, to receive a name ending in `edu`, these entities must have previously registered themselves with `edu`. Thus, the established registration process also serves to limit the number of requests. Furthermore, since FERN's distributed model ensures that registration messages for a group do not propagate outside the group, the registration process is lightweight and remains feasible even at large-scale.

This recursion can be further exploited to provide a distributed security model: suppose a malicious node exists under the `ucsc.edu` domain. If the `ucsc` NRG is able to detect this node, it can protect itself by excising the node from the group. However, this is not the only point of security: even if a malicious node is able to subvert whatever security measures are in place in the `ucsc` NRG, it cannot simply issue requests to any NRG, these requests must still be forwarded through the NRT by `ucsc`. This feature provides multiple failsafes, since any higher-level group (`edu` in this case) identifying malicious activity from one of its member NRGs can protect itself, and the rest of the system, by excising the NRG suspected of being compromised. This stands in stark contrast to the DNS, where servers must fulfill all requests since there is no way of identifying the source of the request.

6.6. Record Integrity And DNSSEC

As mentioned in Section 4.1, a key difference between FERN and DNS is that FERN requires nodes to be a member of an NRG (and, by extension, have a FERN name prefix) before they can resolve names, whereas the DNS serves requests regardless of whether they have a DNS entry or not. Because of this, the DNS security suite (DNSSEC)[24] is designed primarily around authentication and integrity of data, including denial-of-existence, but does not guarantee availability or confidentiality.

With regards to response integrity, FERN is compatible with the tools and protocols of the DNSSEC suite. However, DNSSEC may still be compromised in FERN because it depends on a chain-of-trust model where the identity and record data provided by a DNS zone is verified by its parent zone, all the way up to the root, whose public keys are widely known and distributed. This model depends on the accessibility of the root zone, which is not a problem for the DNS because *all* resolution depends on the availability the root zone. In contrast, the architecture of FERN is designed specifically to remove this dependence: If a member node of `ccrg.ucsc` resolves a name in `inrg.ucsc`, the request only travels as high as `ucsc`.

Since FERN requests travel up the tree to the nearest common ancestor, this introduces a new set of spoofing attacks where a malicious group advertises itself, and nodes wishing to join the legitimate `ucsc` group join the malicious one (presumably of the same name) by mistake. Once joined, the malicious group provides illegitimate responses that appear to be verified with a separate set of keys. However, it should be pointed out that the same threat model exists with DNSSEC today: DNSSEC authentication is performed at the local DNS resolver before the records are delivered to the DNS client initiating the request, so there is nothing in DNSSEC that protects against a malicious local resolver.

This attack-vector highlights a separate part of the existing security model. When clients join a network today, either through WiFi or ethernet, they typically configure their network parameters (such as IP address and subnet) through DHCP. During this process, DHCP also provides the address of one or more local DNS servers for the client to use. FERN uses this same process to bootstrap NRG membership itself, and thus the attack-vector of spoofed NRGs in FERN is equivalent to the attack-vector today of fake or misleading SSIDs (such as the famous “Free Public WiFi” bug in Windows XP [25]). Thus, using DNSSEC with FERN is no less secure than DNSSEC today, since they both depend on legitimate DHCP configuration in the underlay network. This attack-vector merits further examination, especially in the context of social engineering attacks, but this problem is outside the scope of this paper.

7. Testbed Evaluation

The main contribution of FERN is a low-overhead modular framework that enables different systems to interconnect. This modularity makes it hard to compare its performance to existing systems, because any results collected are heavily dependent on the individual groups and their structure. Additionally, all existing approaches today, such as the DNS or UPnP, can function as a FERN group, so they can be considered a subset of FERN functionality.

To confirm the arguments we made in the above sections, and to collect more information about FERN performance, we built a FERN daemon in Java and used it to support three different internal NRG protocols. These protocols, Chord, Server, and Flood, are detailed below, though it is important to note that FERN can support many other forms of communication. We then deployed this daemon on eleven separate nodes located at UCSC, PARC, and UCSB. At each campus we deployed one server connected to the Internet to handle inter-campus queries, at PARC we also used three laptops connected to the campus wireless network, and at UCSC we set up a MANET of four smartphones (and one laptop) running OLSR, with the laptop also connected to the LAN via ethernet.

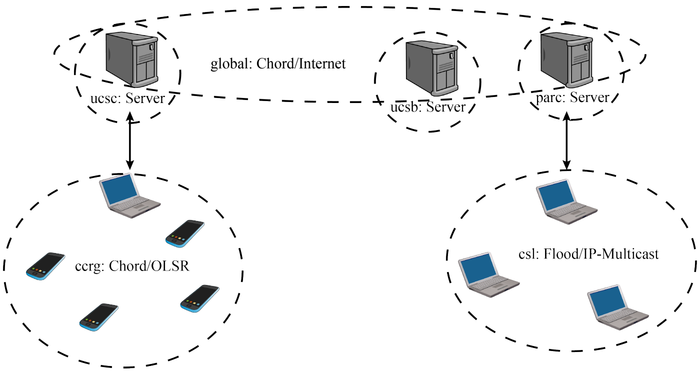

Figure 7: Testbed Implementation

Figure 7 shows how we configured the nodes into the NRT shown in Figure 3. The NRT, network topology, and choice of name resolution protocols was designed to emulate what we envision a wide scale FERN system might look like, with significantly more mobility and dynamic behavior in the lower layers of the NRT than in the upper layers. We ran several name resolution tests using this testbed to evaluate our system and summarize our results below.

Chord: The Chord NRG requires that every node in the group be a member of the Chord DHT[26]. Once a member of Chord, nodes use key-value pairings to map their IP address to any names (nodes or groups) in the NRG that they are responsible for. We use Chord at two levels, shown in Figures 3 and 7, to illustrate its use in different contexts: in the group `ccrg`, we use Chord on top of OLSR to simulate a MANET environment where routing and discovery are done through a DHT[27, 28]. We also use Chord at the highest level, `global`, to emulate an Internet-wide DHT of static name-servers, as discussed in Section 2.3.

Server: The Server NRG very closely resembles a DNS zone. The group consists of one node acting as a name server, and all other nodes in the group are clients of that server. Names and services are registered with this server and resolved by querying the server.

Flood: The Flood NRG is completely decentralized and emulates mDNS. When nodes join a Flood NRG, they subscribe to a multicast address and port combination. Rather than publishing or announcing any names or services, they listen to the multicast address and respond in unicast to any requests for a name or service that they can provide. Conversely, nodes resolve a name or service by simply sending a request to that multicast address. This approach is used at the lowest levels of the hierarchy for two reasons: first, to emulate a MANET running a reactive routing protocol, such as AODV. Second, to illustrate the current role of mDNS and UPnP in today’s home network-

Table 2: Mean Latency of Internal Group Resolution

NRG name	ucsc	parc	global	ccrg	csl
NRG Architecture	Server	Server	Chord	Chord	Flood
Underlay Network	LAN	LAN	WAN	OLSR	IP Multicast
Latency Mean	2ms	3.5ms	180ms	106ms	451ms
Latency Stdev	0.7ms	0.8ms	60ms	21.8ms	173ms

ing environment, where discovery is limited to the reach of the multicast tree.

7.1. Latency

We evaluate latency in our testbed by disabling caching and using tcpdump to time 50 requests internal to each NRG, and 50 requests (each way) between every pair of NRGs. In analyzing these results, we find that the latency of an inter-group request can be expressed as the sum of two main components: the latency of internal group resolution to determine the address of the next hop, and the latency of sending a request or response from one group to another. This distinction is important because it offers greater insight into the system behavior than simply providing end-to-end metrics, especially since resolution times vary dramatically between NRG architectures.

7.1.1. Internal Group Latency

Internal group latency varies tremendously by group, and can dominate the end-to-end latency. As Table 2 shows, in server architectures (including DNS) where the address of the next hop is already known and exists in a table, this time is typically under 5ms, whereas other groups based on DHTs or multicast take significantly longer. Additionally, the groups `csl` and `global` merit additional discussion.

In `csl`, resolution is supported by IP multicast, which results in both a high mean and a significant variance ($\mu = 451$ $\sigma = 173$). Incidentally, these same problems also seen with mDNS. The performance of IP multicast varies significantly with the network topology, traffic load, and even the implementation of the multicast tree. Thus, the results we present in Table 2 do not necessarily reflect performance in other LANs, though they do highlight the fundamental performance problems with using multicast for name resolution.

In our topology, the Chord used for `global` contains three nodes, located at UCSC, UCSB, and PARC. When querying this Chord, we observed latencies ranging from 80 to several hundred milliseconds, depending on the physical location of the data in the Chord. Though our deployment serves as a proof-of-concept, the low number of nodes and their geographical proximity does not accurately reflect the topology of a large-scale Chord. To better reflect a large-scale geographically diverse Chord deployment, we choose to present and compare numbers from the original Chord paper [26] instead, where the authors measured a large 190-node Chord deployed across the entire USA. This topology is closer to what we would expect to see, and therefore their results are a better indicator of latency in this scenario.

7.1.2. Inter-Group Latency

Inter-group latency is determined by two relatively static factors: the physical distance a message must travel (from one node in one group to another node in another group), and the number of times it travels between groups. We compare inter-group latency in FERN to DNS values to get a more accurate understanding of how group hierarchy and structure affects latency. We compare this particular metric to DNS latency because DNS does not have an equivalent internal group component.

Jung et. al. [23] conclude that the number of DNS referrals has a strong effect on the latency of a DNS request. Unfortunately, as examined in Section 6.2, a DNS request with N referrals could potentially result in $2N$ FERN referrals, since it must both climb and descend the tree. However, DNS requests (especially to root and TLD servers) are generally performed iteratively. This distinction is important because the latency of an iterative request with two referrals corresponds to the latency of two complete round-trips from a local DNS server to an authoritative name-server that may or may not be physically close. This problem is highlighted by their KAIST dataset analysis, where they identify a latency “bump” that they correlate with round-trips traversing the Pacific Ocean. Iterative resolution makes this problem worse, since it results in potentially N transpacific round-trips.

In contrast, FERN minimizes inter-group latency by (1) requiring groups to resolve name-requests recursively and (2) organizing nodes in a hierarchy that reflects physical proximity (i.e. assigning countries or physical regions to TLDs). These two concepts combine to forward these requests to their destination and ensure that requests only traverse a particular long-haul link (i.e. the Pacific Ocean) once. With this feature in place, we find that the latency overhead of adding another logical group to the hierarchy is minimal: although it is unlikely that the network address of the group is directly on the route to the target group, with these rules it should be relatively close, and in our tests we find this overhead to typically be under 10ms.

7.2. Caching

Because FERN allows for nodes to be a part of multiple NRGs simultaneously, we augmented the NRT in Figure 3 by registering eight “fake” TLDs of the form {fake1,

Metric	Rules	1	2	3	4	5	Total
Root Group Requests	No Caching	10	10	10	10	10	50
	Ubiquitous Caching	10	9	8	7	6	40
	Caching Rules	10	0	0	0	0	10
Cache Hits	No Caching	0	0	0	0	0	0
	Ubiquitous Caching	0	1	2	3	4	10
	Caching Rules	0	10	10	10	10	40
Cache Entries	No Caching	0	0	0	0	0	0
	Ubiquitous Caching	14	13	13	13	13	66
	Caching Rules	12	1	1	1	1	16

Table 3: Comparison of Aggregated Caching Rules

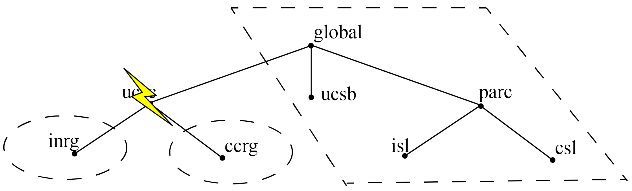

Figure 8: Hierarchy Partitioning Due to NRG Failure

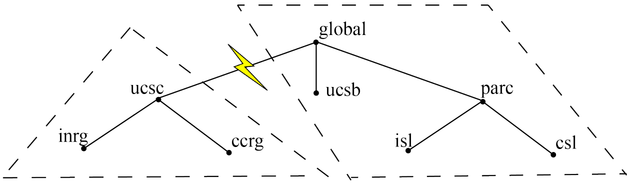

Figure 9: Hierarchy Partitioning Due to Link Failure

fake2, . . . } and adding one entry, `test`, under each fake TLD. We then ran an experiment where five nodes in `ccrg` each resolved one name under each TLD (for a total of 11 names and 55 requests) in our hierarchy three separate times: once with caching disabled, once with all caching enabled, and once with the caching rules in Section 3.8. Table 3 shows our results, represented by three metrics: the number of requests sent to the root group, the number of cache-hits that occurred, and the number of cached entries in the system. In addition to the total values of these metrics, we breakdown the total by node, numbering the nodes 1-5 to show the order they issued their requests in.

The results with no caching serve primarily as a baseline for comparison. Turning on all caching reduces the number of root requests, but only slightly: this is because once a node issues a request that results in a root group query, that node caches the address of the root and then sends all subsequent queries to the root group directly. When an individual node queries the root group directly, it is able to cache the responses, but it cannot share this information with the other nodes. Correspondingly, when we turn on our caching rules we see a tremendous decrease in the number of root queries: the group `ucsc` is the only one that can query the root, and once `ucsc` has a cache entry for every other TLD in the system, there is no need for any further root queries. This benefit is also reflected in the total number of cache entries for the system, which is substantially lower because the TLD entries only exist in `ucsc` instead of being duplicated at each node.

Group:	global	ucsc	ccrg
DNS	0/110	66/110	77/110
FERN	40/110	92/110	77/110

Table 4: Successful Requests With An NRG Failure In The Topology

Link:	parc-global	csl-parc
DNS	30/110	42/110
FERN	50/110	54/110

Table 5: Successful Requests With A Link Failure In The Network

7.3. Fault Tolerance

Given that the topology in Figure 7 consists of 11 nodes, there is a total of 110 different source-destination pairs for a name-request, and in a fully-connected topology they all succeed. However, it is the case that sometimes individual nodes or network links in the system will fail and partition the network into smaller sub-trees. We measure fault tolerance in FERN by introducing two different types of failures into the system and observing how many name-requests succeed; we present these numbers in Tables 4 and 5.

Table 4 examines failures of a DNS zone or FERN NRG. We created this scenario by powering-off all servers for this zone (in DNS) or all members of this NRG (in FERN) and then seeing how many name-requests still succeeded. Figure 8 illustrates the corresponding name-resolution subtrees left in place after a failure of the group `ucsc`.

In contrast, Table 5 considers failures in the underlay network. Here, we left all servers and nodes powered on, but severed *links* in the NRT at the network layer. We did this simply by unplugging ethernet cables or powering off WiFi radios. Figure 9 illustrates the subtrees created by disabling the `ucsc-global` link at the network layer.

As discussed in Section 3.9, FERN groups can contact higher-up groups if necessary. This is part of the key observation that FERN always performs at least as well as DNS with regards to fault tolerance: since FERN "devolves" to a DNS-esque resolution policy when necessary, there can never exist a scenario where DNS resolves a hostname and FERN does not.

Building on this, the data shows that FERN usually outperforms DNS when failures occur, *especially* when the failures occur in higher levels of the hierarchy. This benefit is mainly due to preserving internal connectivity when higher-level groups and links fail, whereas in DNS no-one can resolve any names below a failure in the hierarchy. This is underlined by the complete breakdown of DNS when the root group `global` fails.

8. Conclusions

FERN is novel in its ability to interface radically different name resolution architectures. By providing a unifying framework for these protocols, we have laid a foundation for interoperability between future name resolution protocols that are highly specialized for a particular network environment. Furthermore, we show how to seamlessly extend the current DNS to support FERN-style name resolution.

We have examined and highlighted the differences between FERN and DNS. Our results show that the extra group traversals in FERN do not significantly impact latency, and FERN's forcing recursive queries significantly improves performance. We have discussed the effect of caching and confirmed FERN's fault tolerance and ability to handle network partitions. We have also illustrated FERN's scalability and proved that FERN is deterministic and loop-free.

FERN provides a robust framework for name resolution and service discovery. It provides one global namespace and supports both global and local name resolution, yet does so without the previous constraints on both namespaces. By supporting different name resolution architectures, FERN paves the way for optimization of name resolution protocols for their corresponding networks and serves as an important stepping-stone for interoperability between heterogeneous networks, such as wireless sensor networks and MANETs, home "Internet-of-Things" networks, and the general Internet.

References

[1] S. Cheshire, D. Steinberg, Zero configuration networking: the definitive guide, O'Reilly Media, Inc., 2005.

[2] P. Mockapetris, IETF RFC 1035: Domain names - implementation and specification, IETF Standard.

[3] GoDaddy outage takes down millions of sites, `http://techcrunch.com/2012/09/11/godaddy-says-it-wasnt-anonymous-it-wasnt-a-hack-it-wasnt-a-ddos-it-was-internal-network-issues/`.

[4] Facebook outage blamed on DNS issue, `http://www.itproportal.com/2012/12/12/facebook-outage-blamed-dns-issue/`.

[5] P. Vixie et. al., IETF RFC 2136: Dynamic updates in the domain name system.

[6] Y. Goland et. al., IETF Draft: Simple service discovery protocol.

[7] E. Guttman, J. Veizades, IETF RFC 2608: Service location protocol.

[8] R. Droms, IETF RFC 2131: Dynamic host configuration protocol.

[9] S. Cheshire, B. Aboba, E. Guttman, IETF RFC 3927: Dynamic configuration of IPv4 link-local addresses.

[10] R. Cox, A. Muthitacharoen, R. T. Morris, Serving DNS using a peer-to-peer lookup service, in: Peer-to-Peer Systems, Springer, 2002, pp. 155–165.

[11] V. Pappas, D. Massey, A. Terzis, L. Zhang, A comparative study of the DNS design with DHT-based alternatives, in: Proc. INFOCOM, 2006.

[12] Y. Song, K. Koyanagi, Study on a hybrid P2P based DNS, in: Proc. CSAE, 2011.

[13] T. Vu, A. Baid, Y. Zhang, T. D. Nguyen, J. Fukuyama, R. P. Martin, D. Raychaudhuri, DMap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet, in: Proc. ICDCS, 2012.

[14] K. V. Katsaros, N. Fotiou, X. Vasilakos, C. N. Ververidis, C. Tsilopoulos, G. Xylomenos, G. C. Polyzos, On inter-domain name resolution for information-centric networks, in: NETWORKING 2012, Springer, 2012, pp. 13–26.

[15] A. Venkataramani, A. Sharma, X. Tie, H. Uppal, D. Westbrook, J. Kurose, D. Raychaudhuri, Design requirements of a global name service for a mobility-centric, trustworthy internetwork, in: Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on, IEEE, 2013, pp. 1–9.

[16] U. C. Kozat, L. Tassiulas, Service discovery in mobile ad hoc networks: An overall perspective on architectural choices and network layer support issues, Ad Hoc Networks 2 (1) (2004) 23–44.

[17] M. D'Ambrosio, C. Dannewitz, H. Karl, V. Vercellone, MDHT: a hierarchical name resolution service for information-centric networks, in: Proc. SIGCOMM workshop on ICN, 2011.

[18] P. Grace, G. Blair, S. Samuel, ReMMoC: A reflective middleware to support mobile client interoperability, in: Proc. DOA, 2003.

[19] A. Friday, N. Davies, N. Wallbank, E. Catterall, S. Pink, Supporting service discovery, querying and interaction in ubiquitous computing environments, Wireless Networks 10 (6) (2004) 631–641.

[20] J. Allard, V. Chinta, S. Gundala, G. Richard III, Jini meets UPnP: An architecture for Jini/UPnP interoperability, in: Proc. Symposium on Applications and the Internet, 2003.

[21] J. Crowcroft et. al., Plutarch: An argument for network pluralism, in: Proc. FDNA, 2003.

[22] V. Ramasubramanian, E. G. Sirer, The design and implementation of a next generation name service for the internet, in: Proc. SIGCOMM, 2004.

[23] J. Jung, E. Sit, H. Balakrishnan, R. Morris, DNS performance and the effectiveness of caching, Transactions on Networking 10 (5) (2002) 589–603.

[24] S. Weiler, D. Blacka, IETF RFC 6840: Clarifications and implementation notes for DNS security (DNSSEC).

[25] The zombie network: Beware 'free public wifi', `http://www.npr.org/2010/10/09/130451369/the-zombie-network-beware-free-public-wifi`.

[26] I. Stoica et. al., Chord: a scalable peer-to-peer lookup protocol for Internet applications, Transactions on Networking 11 (1) (2003) 17–32.

[27] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, A. Rowstron, Virtual ring routing: Network routing inspired by DHTs, in: ACM SIGCOMM Computer Communication Review, Vol. 36, ACM, 2006, pp. 351–362.

[28] D. Sampath, J. Garcia-Luna-Aceves, Scalable integrated routing using prefix labels and distributed hash tables for MANETs, in: Proc. MASS, 2009.