Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2016-012 September 26, 2016

Examining Key Mobility Resources through Denial of Service Attacks on proposed Global Name Resolution Services

Colleen T. Rock

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

CSAIL

Examining Key Mobility Resources through Denial of Service Attacks on proposed Global Name Resolution Services

by Colleen T. Rock

S.B., Massachusetts Institute of Technology (2015),
Computer Science and Engineering

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Computer Science and Engineering
at the
Massachusetts Institute of Technology
September 2016

Author: ____
Department of Electrical Engineering and Computer Science
September 5, 2016

Certified by: ____
Dr. Karen R. Sollins
Principal Scientist
MIT Computer Science and Artificial Intelligence Laboratory

Accepted by: ____
Dr. Christopher Terman
Chairman, Masters of Engineering Thesis Committee

Abstract

The problem we address in this thesis is to uncover the design elements in a network architecture design that may open it up to denial of service (DoS) attacks and to expose the tradeoffs in mitigating those DoS opportunities. We take as our candidate network architecture design the Future Internet Architecture project **MobilityFirst**. MobilityFirst's overarching goal, driven by increasingly available wireless communication, is the support of mobility in an Internet architecture. At its core, MobilityFirst separates identification from location, as distinct from the current Internet architecture, and postulates the existence of globally unique, flat identifiers. In order to support mobility in this context, it also postulates a global name resolution service (GNRS). In this thesis we examine three alternative designs for the GNRS and the opportunities they expose for DoS attacks. We consider each one in depth analytically. As an example, we then study one particular attack in depth and are forced to conclude that approaches to mitigating this attack would have significant negative impact on the support of mobility thus exposing the dilemma in such system design tradeoffs.

Acknowledgements

I would like to thank my advisor, Karen Sollins, for all of her guidance and support. Additionally, I want to acknowledge my parents for their unwavering support of me and my education, my brother, Dan, who sympathized for my research struggles, and all of my friends for their encouragement, especially those who are part of the Internet Policy Research Initiative at MIT.

This work was funded in part under NSF Grant 1413973, "NeTS: Large: Collaborative Research: Location-Independent Networks: Evaluation Strategies and Studies".

Table of Contents

1. Introduction

1.1 Example

Imagine a post office that provides a service allowing you to send them a postcard saying "I'm staying at my friend's house for a couple days; please send my mail there." This allows you to continue receiving your mail while couch-surfing. However, this service that supports frequent movement requires the post office to keep track of all the places where people request that their mail be sent. What if someone mischievously sends many postcards falsely claiming that they moved, or impersonates you saying that you moved? The post office now has to sort through all of these post cards to determine which ones are valid, and although the postman could check to see if you are actually at the new location while delivering mail there, this verification would be very time consuming.

1.2 Summary of the Problem

The example[1] in the prior section illustrates some of the issues that arise when a system attempts to support mobility occurring at a time scale comparable to information delivery. In the Internet, unlike the postal system, your name and address are represented as one value, your Internet Protocol (IP) Address. This is referred to as conflating name and location, which are actually two different things. A unique identifier acts as a "name" for each device connected to the Internet, which doesn't change throughout the lifetime of the device, whereas your cell phone's network address can change from Verizon LTE to Wi-Fi mid download. If you switch networks in the current Internet, your phone changes IP addresses and cannot pick up its conversation with the server where it left off. If all the server knows about a device is its IP

[1] Based on the Mobile Internet Protocol where the post office represents a home agent.

address, then when your phone switches networks the server has no way to know it's talking to the same phone. Instead, the phone has to reestablish a connection with the server and try the download again.

MobilityFirst (MF) is a design for a Future Internet Architecture to replace the Internet. It supports separating a name, what they call a Globally Unique Identifier (GUID), from a device's location, called a Network Address. Once names are separate from locations, your phone could switch networks but continue its conversation, or session, with the server, because your phone retains its GUID, say G_P. The server is still sending to phone G_P, and your phone can verify that it is G_P, it just happens to be at a different location now. Although MobilityFirst creates a desirable behavior allowing your device to move between networks without re-starting its remote sessions, just like the post office, this new feature may make the system of delivering Internet traffic susceptible to attack in new ways. This thesis studies the new system that supports this feature, the Global Name Resolution System.

1.3 Summary of Findings

Since MobilityFirst is a mostly untested, new architecture, we aim to understand its capabilities, strengths and weaknesses. In this thesis we examine this architecture more deeply through the "lens" of denial of service opportunities that arise from its unique characteristics intended to support wide scale and extremely dynamic mobility. The purpose of studying mobility is to unearth more about the resources that are critical to node mobility, and gain insight informing how to design robust mobile protocols in the current Internet and systematic ways of thinking about mobility.

While many different attacks are discussed, our main finding involves a malicious server attacking a target server in a distributed hash table implementation of a Global Name Resolution Service. In this case, solutions to address the attack would involve tradeoffs that negatively impact the system's ability to support rapid mobility, limiting either which GUIDs could register with a part of a distributed GNRS or how frequently they could change their name record.

This thesis presents related work in Chapter 2 to set the background for Global Name Resolution System Designs, discussed in Chapter 3. This is followed by a discussion of their key resources and possible vulnerabilities in Chapter 4 and a demonstration of an attack in Chapter 5. Chapter 6 discusses the effects of the underlying transport layer protocol. Chapter 7 overviews possible future work and Chapter 8 concludes the thesis.

2. Background

Using Denial of Service to study the efficacy of MobilityFirst is a part of Sollins's work proposed in [1]. Other work has been done studying Denial of Service attacks in another Future Internet Architecture, Named Data Networking, but not in MobilityFirst. Their examination of Denial of Service attacks led to a better understanding of Named Data Networking architecture and influenced a revision of the protocol to include negative acknowledgements [2, 3, 4]. A well-known attack in Named Data Networking is an Interest Flooding attack, where an attacker floods a router's Pending Interest Table with many requests for nonexistent named content. Since the content did not exist, these requests would remain in a routers Pending Interest Table, filling it up. The negative acknowledgement implements the software design paradigm of "failing fast", and informs the router that it can drop these requests because the data is not coming, and significantly decreases the effectiveness of the attack.

2.1 MobilityFirst

MobilityFirst (MF) [5] is a Future Internet Architecture proposed by Venkataramani et al. that assumed we could replace the Transport and Internet Layers of the Internet protocol suite. MobilityFirst's primary design goals are mobility and trustworthiness.

The main aspects of the MobilityFirst architecture are storage aware routing, the management plane, the compute layer, and the Global Name Service (GNS). The management plane is federated and provides aggregate network information used to perform accounting and address problems and attacks. The GNS consists of a Name Certification Service and a name resolution service, often called a Global Name Resolution Service (GNRS).

This thesis focuses on the GNRS, which maps GUIDs to NAs. Instead of IP addresses, MobilityFirst has Globally Unique Identifiers (GUIDs) which are like “names”. Like other proposed Future Internet Architectures, MF does not conflate identity and location [5, 6]. In MF, GUIDs are self-certifying and represent interfaces, devices, services, content, human end-users, and groups of GUIDs [5]. Locations are represented by Network Addresses. A Network Address is the identifier of a network. Unlike an IP address, it is not unique to the named entity, often a device, that is attached to it.

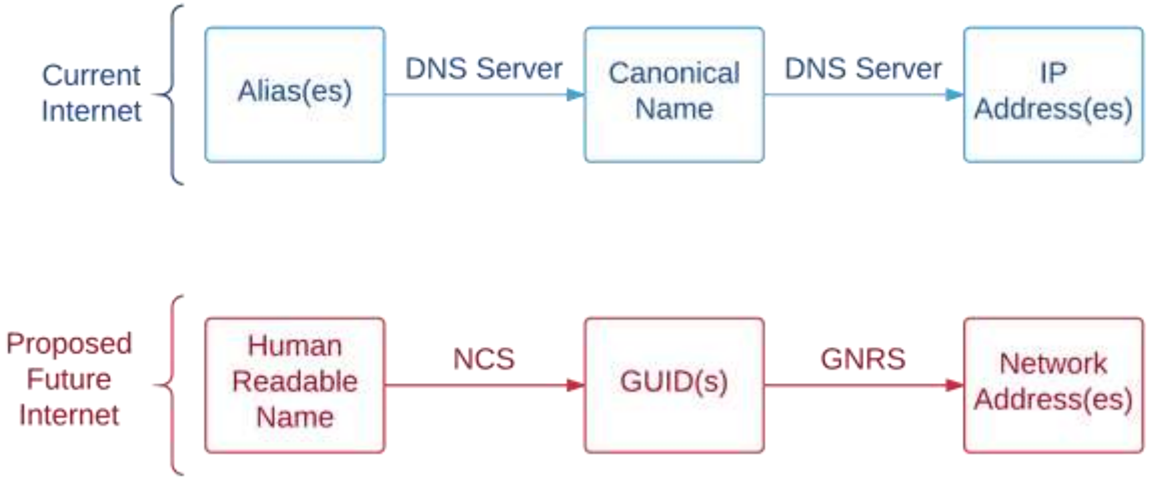

*Figure 1. **GNRS and DNS.** The GNRS provides one of the two resolution tasks that the Domain Name Service handles in the current Internet. The other is performed by a Name Certification Service. Recall from 2.1 that a Network Address is a reference to a sub-network that knows where a GUID is. NAs are used inside the Future Internet to forward traffic to its destination.*

MF encourages competition between Name Certification Services (NCSs), which bind “human-readable descriptors” to GUIDs. Unlike IP addresses, GUIDs are not hierarchical. Since they are flat, naively, looking up the location of a GUID can take time on the order of the number of GUIDs. Since routing at the level of GUIDs can be so inefficient, a Global Name Resolution Service (GNRS) is necessary to efficiently map GUIDs to one or more Network Addresses.

Network Addresses can be thought of as a logical gateway that corresponds to a subnetwork, either an Autonomous System or Internet Service Provider, which contains GUIDs. Network Addresses are also self-certifying. Another key resource is the GUID and its self-certifying nature. The GNRS essentially *resolves* a GUID by providing a translation from GUID to NA. We are looking at the GNRS specifically because it is a key resource everything needs access to: "The GUID-based communication assisted by the GNS forms the 'narrow waist' of the MF architecture…" [5].

2.2 Denial of Service Attacks

Denial of Service (DoS) attacks aim to deny access to a service or part of a network to an audience. A Distributed Denial of Service (DDoS) attack is a DoS attack that is distributed across many computers, often a recruited botnet. DoS and DDoS attacks illuminate the security and resiliency of proposed architectures by demonstrating how a system can cease functioning [1]. The challenge of trying to prevent an attack without creating an opportunity for a new one can be especially insightful.

Mirkovic et al. suggest ways to group DDoS attacks: by degree of automation, source address validity, attack rate dynamics and persistence of agent set, possibility of characterization, exploited weakness, victim type, and the impact on the victim [7]. In their "Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems" Peng et. al. describe four categories of defense against DoS: Prevention, Detection, Source Identification, and Reaction [8]. Prevention mainly uses filtering to limit the ability to spoof IP addresses, including ingress/egress filtering, Router-Based Packet Filtering which basically filters between ASs at border routers, and the Source Address Validity Enforcement (SAVE) Protocol, in which routers keep a table with valid addresses for each interface. The authors ultimately claim that spoofing is happening less frequently in DoS attacks. The path of least resistance can be to find vulnerable systems on the Internet and use their IP addresses, as in a botnet. One reason that DDoS attacks are so hard to detect is that a "flash crowd", legitimate traffic that can occur when many people suddenly look up the same thing, can appear like a DDoS attack. Denial of Service attacks are tricky precisely because they can look like legitimate traffic; it's much easier to have false positives than other network security attack types. One could even argue that an "attacker" trying to take advantage of the system still falls under legitimate use if they are following the system's protocol. If not, in the worst case scenario where an attacker can generate false positives that affect their target, stopping what looks like attack traffic actually helps the attacker achieve their goal.

Defenses for attack detection break into two main categories: DoS attack-specific, which look for known attacks, and anomaly-based, where a traffic is compared to a "normal profile". The challenge with identifying the source of an attack is that IP routing is stateless and addresses are not verified before they are allowed to send traffic. One suggestion to aid in detection is "Probabilistic IP Traceback" where routers probabilistically append their IP address to packets such that the packets can be "traced back" from where they came from. Overall, Peng et. al. conclude that the decentralization of the Internet and lack of economic incentives to implement defenses, along with the current state of affairs being "good enough" are the main challenges that prevent the implementation of defenses. They suggest that ISPs are a good place to start because they are a preexisting *group* of routers than can work cooperatively without overhauling the entire Internet.

A significant feature of both Mirkovic's and Peng's papers is that they focus on types of attacks. In contrast, this research will focus on the kinds of resources that are attackable, in order to understand how those resources are utilized to support supporting rapid mobility in MobilityFirst.

3. GNRS System Designs

The Internet was not designed for mobility, but with the development and prevalence of wireless technology, mobility has become commonplace. MobilityFirst was designed with the primary goals of mobility and trust. Mobility occurs when a single identity relocates. This includes, but does not always necessitate, a device physically moving. For example, a phone switching between WiFi and 4G LTE changes the network address of the phone. Even though the phone did not physically move, the route a packet takes over the network topology may have changed drastically, resulting in packet loss due to mobility. This mobility can occur rapidly, causing Future Internet Architectures to support the separation of nominal identifiers from location identifiers. As Vu et. al. stated in their paper on DMap, one of the proposed GNRS implementations: “Intuitively, it is easier to work with networking primitives based on identifiers when the locator changes faster than the timescales of the communication session” [9, p. 1]. This is true because the details of mobility and how addresses change with mobility are abstracted away by the GUID.

Just like the white pages allow us to look up someone’s address given their name, the Domain Name Service will translate well-known names, like “google.com” into Google’s IP address for us. Since IP addresses represent both name and location, only one step is necessary. In MobilityFirst, after translating a human-readable name like “google.com” to a corresponding GUID, there is an extra step to find the location of that GUID. This is advantageous because it allows an entity to retain its globally unique identifier, but change its location. This extra step of resolving a GUID to its current location is accomplished by the Global Name Resolution Service. The first step, resolving a human-friendly name to a GUID, is performed by a Name Certification

Service. The details related to this resolution are outside the scope of this project because it is outside the core architecture of MobilityFirst.

This thesis focuses on the second step of retrieving a Network Address for a GUID, which is done with a Global Name Resolution Service (GNRS). Adding this location flexibility creates new opportunities for attacks in the global network, as adding new features creates opportunities for bugs and misuse in any system. In the most basic form, there are three primary attackable resources in a network: bandwidth, computation, and storage. The proposed ideas have a tradeoff between these three resources. For example, supporting mobility without having a physically separate GNRS would utilize shared storage with routers inside the network to obtain and store up to date location information. This is the case in two GNRS designs: DMap and GMap. If the location information is represented in a resolution service that is invoked prior to the traffic being sent, then the storage, computation and bandwidth resources proposed for the resolution service must be analyzed to determine how they introduce vulnerabilities into the network.

There are three proposed GNRS implementations, DMap, GMap and Auspice. All of them support the ideas of Insert and Lookup Requests. Insert Requests are messages designed to support the insertion of a mapping from a GUID to its Network Address(es) into the GNRS. This mapping, which may hold additional data, is called a name record, and the Network Address(es) may be called the bindings of that GUID. Additionally, the GNRSs support Lookup Requests that fetch the name record. The rest of this Chapter describes and compares the three implementations. Their DoS vulnerabilities are discussed in Chapter 4.

3.1 DMap

The first proposed GNRS implementation, DMap, named after its use of Direct Mapping, is an in-network distributed hash table[2] [9]. DMap distributes *K* global replicas for each GUID to NA mapping among participating routers. When a new device connects, its GUID is hashed into *K* IP addresses using *K* independent hash functions. The servers that announce these IP addresses store a mapping between the GUID and its location(s). The location, a Network Address, is a reference to the Autonomous System that contains the new device. An example with *K*=1 is shown in Figure 2.

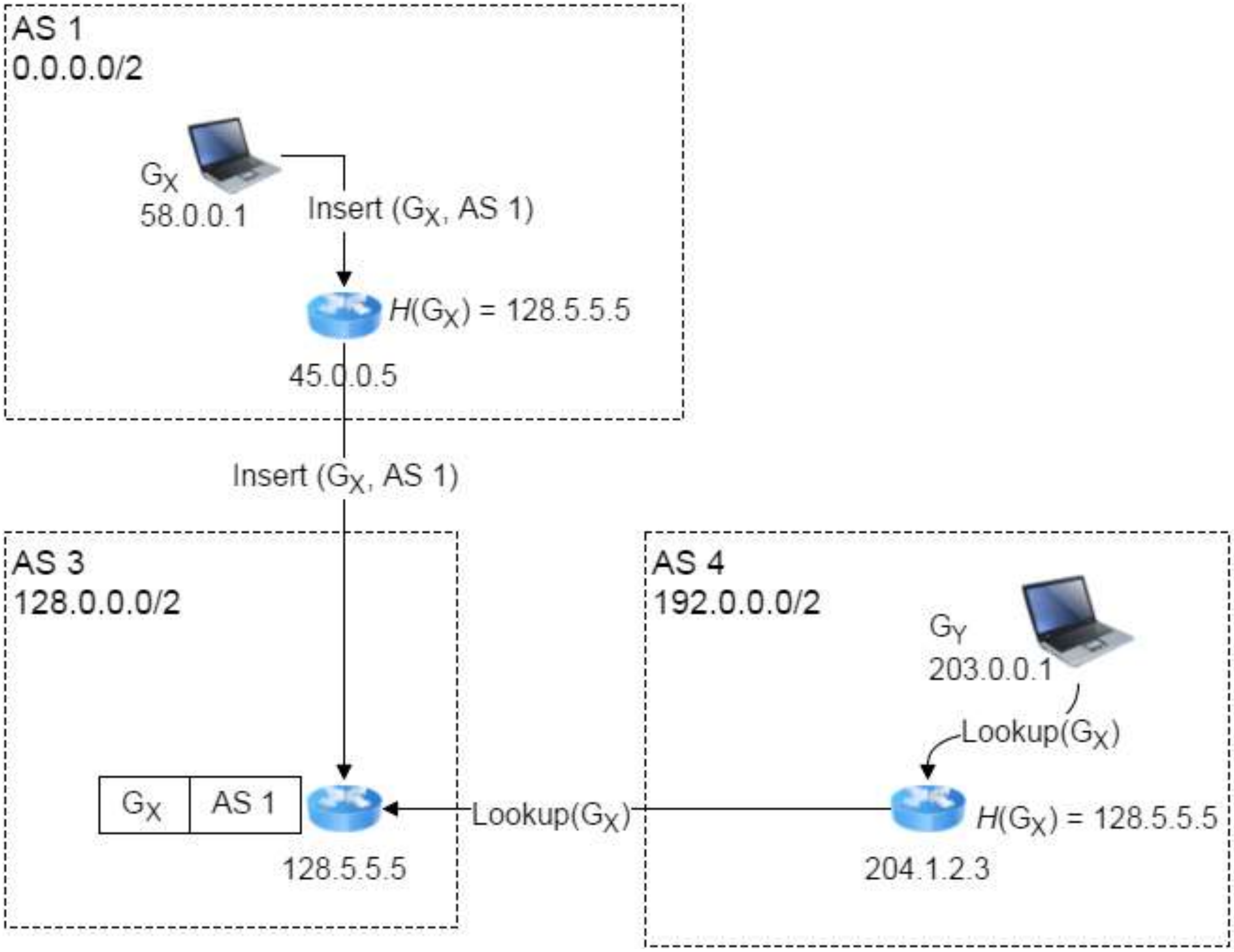

Figure 2. ***DMap Insert and Lookup****. GUID G_X, located at 58.0.0.1, connects to the network via the server at 45.0.0.5. This server hashes G_X with hash function H to obtain the address of the server that will store G_X's location, in this case, 128.5.5.5. G_X's location can be queried from 128.5.5.5, which any server can compute by taking $H(G_X)$.*

[2] For background information on distributed hash tables, see Chord [30].

3.1.1 Local Replication

In Figure 2, if a different router in AS 1 wants to route to G_X, it would first have to hash G_X: $H(G_X) = 128.5.5.5$. Then, if naively following the DMap protocol, it would ask 128.5.5.5 where G_X is located. This could result in an unnecessarily long lookup time, especially if AS 1 and AS 2 are far away or do not have a high bandwidth connection. DMap attempts to deal with this by including local replication. In addition to each of the *K* global copies of the (G_X, AS 1) mapping, it will also use a hash function to determine a local server on which to store (G_X, AS 1). When servers look up a GUID, they simultaneously send requests to the local server that would know if the GUID is in their Autonomous System and to a “global” copy, using one of the original *K* hash functions. As shown in Figure 3, the server in AS 4 also queries 198.4.4.4 to see if G_X is located in AS 4.

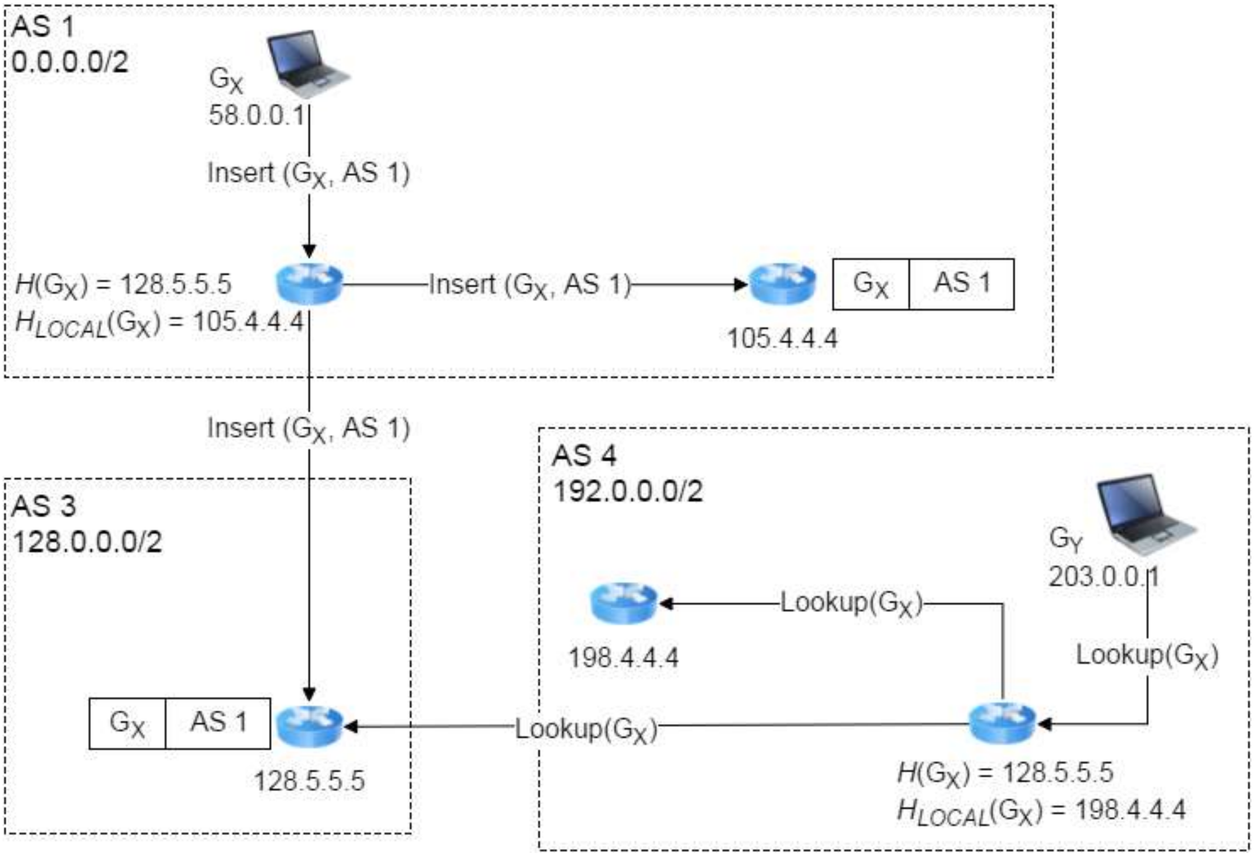

*Figure 3. **DMap with Local Replication**. When looking up a GUID, servers use a local hash function, H_{LOCAL}, to simultaneously ask a local server if G_X is in the same Autonomous System as the requester. Similarly, when a GUID is inserted, a local hash function is used to place a DMap record in that GUID's AS.*

3.1.2 Multihoming

DMap supports multihoming, a device attached to the network with more than one connection. Multiple locations can be sent with an original Insert request. Alternatively, if DMap has already has a location for a GUID and receives another Insert message, the additional address is simply appended to the list of addresses for that GUID. DMap permits each GUID to be associated with up to 5 NAs at once, and to assign expiration times and prioritization weights to each network address. Since a new Insert message does not remove an old network address for a GUID, an Update Message is also defined for when the GUID moves from a network address to a new one.

3.1.3 Edge Cases

Like many implementations of complex protocols and systems, DMap defines and handles some uncommon but expected edge cases. Since they are expected to be outside the norm of operation, edge cases often lead to interesting attack opportunities. An edge case DMap considers is that of hashing GUIDs to "nonexistent" IP addresses. DMap is currently implemented on top of the IP layer, and relies on IPv4 [10]. DMap uses the list of BGP prefix announcements to know which ASs announce which subsets of the IPv4 space in the current version of DMap. The IP Hole Problem is the phenomenon that some IP addresses are unclaimed by ASs. If a GUID hashes to an unclaimed network address, it is rehashed up to *M*-1 times. If, after *M* total hashes, a GUID still hashes to an unannounced IP address, a Deputy AS, an AS with the closest IP distance to the final hashed IP address, will host the GUID-NA mapping. Later, if an AS has new servers join and announces additional IP addresses, it is possible that some already hosted GUID mappings would belong on the new server. The DMap architecture includes a *GUID migration message* so that the new announcing AS can tell the corresponding Deputy AS that it has now joined the network and can host the mapping, prompting the Deputy AS to drop that mapping.

3.2 Auspice

Auspice [11, 12] is another proposed global name resolution service (GNRS) that can support current Internet architecture and MobilityFirst Future Internet Architecture. Auspice's records are more broadly defined than DMap's GUID to NA mapping. They allow each GUID to be associated with a name record consisting of arbitrarily many key value pairs. This supports "novel network-layer functions such as simultaneous mid-connection mobility and context-aware communication" [11, p. 7]. Auspice allows blacklists, which prohibit write access to

GUIDs listed, and/or whitelists, which only allow write access to GUIDs listed, for each key value pair in a GUID's name record. Auspice defaults to only allowing the GUID itself to change its Network Address(es), but may allow a third party system to append traits, like geo-location or friends, to a GUID's name record. An example Auspice name record is shown in Figure 4.

```
{"occupation":"rocket scientist", "ip address":"127.0.0.1", "name":"frank",
"location":"work", "friends":["Joe","Sam","Billy"],
"flapjack":{"sally":{"left":"eight","right":"seven"}, "sammy":["One","Ready","Frap"]}}
```

*Figure 4. **Example Auspice Name Record**. Name records are JSON objects. This is an example from edu.umass.cs.gnsclient.examples.ClientExample in [13].*

3.2.1 Replica-Controllers

Auspice uses a fixed number of *replica-controllers* for each name, which "dispatch" active replicas for that name. The replica-controllers collect aggregate frequency and location information about requests for their assigned GUID made to its active replicas, and control the placement of active replicas. As shown in Figure 5, active replicas actually store the GUID and its name record, including its network address. A client can determine replica-controllers for a GUID using a hash function, similar to lookups in DMap. After sending a request to a replica-controller, the client receives a list of the current replicas for that GUID and can ask the nearest replica for the GUID's name record.

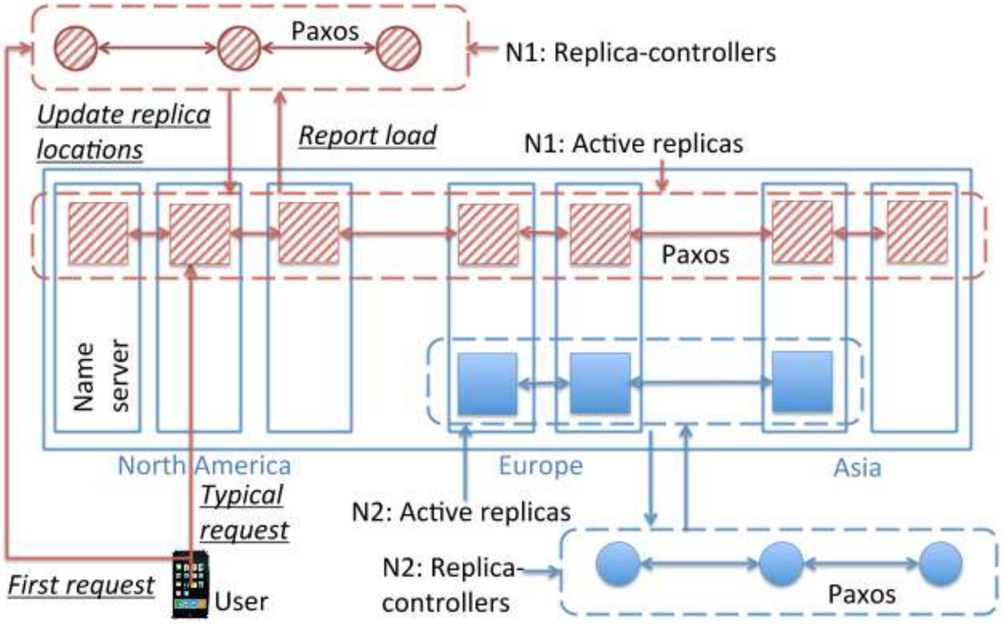

Figure 5. ***Auspice System Overview****. "Geo-distributed name servers in Auspice. Replica-controllers (logically separate from active replicas) decide placement of active replicas and active replicas handle requests from end-users. N1 is a globally popular name and is replicated globally; name N2 is popular in select regions and is replicated in those regions" Figure and Caption from [11, p. 5].*

Replica-controllers are contacted "infrequently" by a client or Local Name Server (1) when the Local Name Server (LNS) receives a request for a name it has not seen before, and (2) when the active replica the LNS had previously queried is no longer an active replica for a specific name. The assumption that this is infrequent is discussed in Chapter 4.

3.2.2 Demand Aware Placement

Auspice places replicas with the key, value pairs considering both recent demand and update frequency. To keep the "demand aware" placement recent, the placement of replicas is evaluated and adjusted after a pre-determined time period called an epoch. At the beginning of each epoch, Auspice tunes the number and location of active replicas for each GUID using Equation 1. The system considers the update and lookup rate for that GUID, the geo-distribution of requests for that GUID, and the overall load on the system (across all GUIDs).

$$\text{number of replicas for GUID } G_X = M + \beta\left(\frac{G_X\ lookup\ rate}{G_X\ update\ rate}\right)$$

$$\text{where } \beta \text{ satisfies } \mu C = \sum_{i\,\in\,\text{GUIDs}} M(i\ update\ rate) + (\beta + 1)(i\ lookup\ rate)$$

Equation 1. ***Demand-Aware Replica Placement.*** *C is the total capacity of all servers running Auspice. μ is a parameter less than one representing the target utilization. M is a fault tolerance parameter indicating the minimum number of replicas for a GUID. This equation is a reformatted version of Equation 1 in [11, 12].*

3.2.3 Paxos

Auspice allows a GUID to recursively map to a list of other GUIDs, so that an application can send to multiple GUIDs in a category the same way it would send to a single GUID. An often-used geocasting example is sending a message to all taxis in New York City. In order to have the same behavior no matter which replica a message is sent to, GUID lists must be synchronized. In order to keep these operations consistent across the distributed system, Auspice uses Paxos. There are two Paxos instances for each GUID, one for the Replica-controllers and one for the Active Replicas. Total write ordering is guaranteed across updates for active replicas. Stoppable Paxos is used so that when the active replicas are updated, at most once each epoch the Paxos instances can be paused until the switch has been made. The authors of Auspice have designed and implemented their own version of Paxos for Auspice, which they call GigaPaxos [14].

3.3 GMap

GMap is an updated version of DMap that also considers geographic location when distributing the GUID-NA pairs [15]. GNRS Servers are also represented by GUIDs, which are called Server Identifiers (SIDs). GMap distributes *K* replicas for each GUID to NA mapping among global replicas, regional replicas and local replicas. Regional replicas are in the same country as the GUID's NA and local replicas are in the same city or metropolitan area as the GUID's NA. Assuming the geolocation of the NA and the replicas are known, there is no extra information that must be stored in order to place the GUID-NA mappings at replicas. However, it is unclear what GMap does when a GUID is multi-homed. GMap's authors argue that updating per GUID as in Auspice leads to scalability problems.

GMap does not have the same extensibility design goals as Auspice, but does mention that "Application-specific policies on the GUID-to-NA mapping are supported by extensible fields of key, value pairs" [15, p. 6]. Unlike Auspice, GMap delegates the computationally expensive task of searching for GUIDs by attributes to the Name Certification Service, although recursively defined groups of GUIDs are still handled by the GNRS. By leaving the computational demands in the GNRS, an attacker wishing to flood the network with nonsense packets still has a multiplicative attack opportunity. The attacker can send a request to a GUID that recursively resolves to many GUIDs and have GMap send its packet to each of them.

In Auspice, a server needs to ask a replica-controller which replica is responsible for any given GUID. In GMap, instead of needing to ask a central resource, each server can compute the SID responsible for a specific GUID. The basic idea is similar to DMap, though the function used to determine the server differs. The authoritative SID for a GUID is defined to be the SID that, when XORed with the GUID, yields the smallest value.

3.3.1 Asking the Geographically Closest Server

It makes sense to ask a nearby server where a GUID is, if any nearby servers know. Since there is no way to know a priori whether GUID X is in the current region – we are looking up its location after all – the requesting server assumes it is and asks a server that would be X's local replica. Computing which server to request the mapping for a GUID becomes complicated when trying to leverage geographic location. When a router receives a lookup request for X with GUID

G_X, it first checks its cache for G_X. On a cache miss, the router will compute all of the replicas for G_X, assuming that X is in the same local area as itself. If X is not in the same local area as the router, then the router will not receive a response from any of the K_3 servers in its city whose SIDs XORed with G_X have the lowest values. The server will then send requests to the K_2 servers in its region whose SIDs XORed with G_X have the lowest values, which will only respond if X is in the same region as the requesting server. Finally, if X is not in the same region as the requesting server, the server will ask the global replica(s) for X, which are not constrained by any geographic boundary and have the lowest value when XORed with G_X. This situation is illustrated in Figure 6.

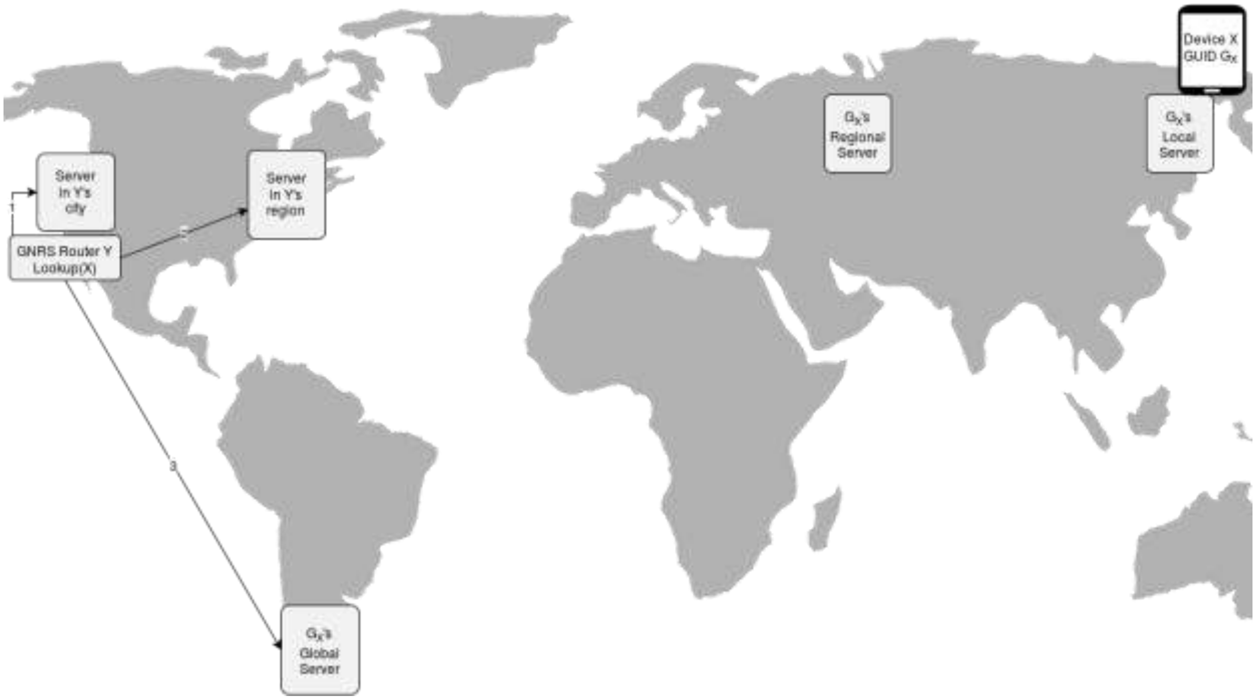

Figure 6. ***GMap Lookup Requests***. *Router Y looks up the NA of GUID G_X. It first contacts two servers that do not have any information for GUID G_X, since it incorrectly assumes X is in the same city, then region, as Y.*

Since the global replica(s) are not dependent on X's actual location, the computed global replica will always be the actual global replica for G_X, and can be considered an "authoritative" server for G_X. When X moves enough to change its region, the K_1 authoritative server(s) will still be the nearest XOR to G_X, and will just have to be updated, but the K_3 local and K_2 regional servers will need to discard X's information and the new local and regional replicas will have to be determined and updated. If global, regional, and local replicas are distributed equally, two-thirds of the servers change when X changes regions. Whenever X moves, the cached copies of G_X-NA_X become stale.

3.3.2 GMap Uses Caching to Compensate for Popular GUIDs

In GMap, caching is used to spread out the workload associated with looking up "hotspot GUIDs" [15] by keeping additional copies of popular GUID's name records in routers' caches along the paths request responses traverse. The contents of a GMap cache entry are shown in Figure 7.

GUID → NA	
Remaining TTL	
P_t	Go-through probability for time period t
U_t	Number of NA updates during t
H_t	Number of hits for GUID during t
C_t	Perceived update counter $C_t = U_t / H_t$
C_{t-1}	Perceived update counter for last time period

Figure 7. ***GMap Cache Entry***

$$P_{t+1} = P_t + (C_t - C_{t-1}), \text{ constrained to } [0, 1]$$

Equation 2. ***GMap Cache Go-through probability****. The go-through probability dictates the frequency with which a cache hit is allowed to "go-through" to the next hop. Allowing some traffic through passively updates the cache entry, keeping it current.*

The caching in a GMap server takes into account the local demand for each GUID in the server's cache. It does not collect aggregate information and statistics about every GUID, and since hits and updates are only recorded and used locally, geographic information is not stored. A server is storing a limited amount of load information for each GUID: keeping two counters, U_t and H_t, and two static values, P_t and C_{t-1}.

3.4 Summary

Separate from a Name Certification Service, the main task of a GNRS is to retrieve Network Addresses (NAs) associated with globally unique identifiers (GUIDs).

The proposed Global Name Resolution Services make tradeoffs. DMap is the simplest and most straightforward. In DMap and GMap, there are a fixed number of replicas for each GUID, regardless if it someone's personal device or a server with much more traffic. GMap uses demand-aware caching to increase the number of copies of mappings for popular "hotspot" GUIDs without keeping global geo-specific statistics on each GUID.

Since Auspice allows recursive GUIDs, it uses a version of Paxos to guarantee consistency. It is unclear how GMap keeps a consistent view of recursively defined GUIDs. DMap makes a best-effort attempt to return the correct list of network addresses for a GUID and does not concern itself with consistency. If a network address does not actually have the desired GUID, the client can request the mapping again. In the current implementation, a random server containing the desired mapping is queried; with more than two replicas, it is likely that a different server will be queried if the mapping is requested again.

There are some cases where it makes sense for a multi-homed device to have the priorities associated with its NAs differ based on the location of the mapping. For example, consider a device multi-homed on the MIT network and a Comcast network. This device may want to be reached by other computers on the MIT network through its MIT network address, and by its

Comcast network address by devices external to MIT. The GNRS could support this by allowing any GNRS servers serving the MIT network to denote a higher priority for the MIT network address, and all other GNRS servers expressing a higher priority for the Comcast address. Similar examples arise for devices connected to a military and non-military network or company internal and an external network. Additionally, this type of behavior is useful for load balancing. For example, in the current Internet, routing policies and priorities differing among different parts of the network is currently supported in the BIND implementation of DNS Internet protocols [16].

None of the GNRS designs currently support this, though some are more adaptable to do so than others. Although Auspice may be best set up to accurately predict which network address may be the most efficient in each location because of its vast collection of statistics, it forces consistency and guarantees that the name record obtained from different active replicas is the same, preventing customization of name records on different active replicas. DMap allows the GUID to set the priorities for different network addresses in the insert message, which is forwarded to the rest of the *K* servers, so the name records should be identical, but it is not guaranteed. Since DMap already uses a separate hash function for storing the GUID's record locally within its AS, it would be feasible to have that record be distinct from the *K* other copies. GMap is perhaps in the best position to customize name records based on their location, since it already runs different hash functions for each locality (local, regional and global).

	Auspice	**GMap**	**DMap**
Location relative to network	Overlaid on top of network	In-network	In-network
Algorithm Type	Replicated State Machine	Distributed Hash Table	Distributed Hash Table
Record Content	GUID to arbitrarily many key value pairs	GUID to NA(s); GUIDs may be recursively defined	GUID to up to 5 NAs, each with an expiration time and prioritization weight
Replica Placement	Geo-located based on requests	Geo-located based on GUID's physical location	Not Geo-located. One Replica in the GUID's AS
Number of Replicas	Adjusts # of replicas for each GUID based on recent demand and update frequency	Fixed # of replicas for each GUID (each GUID has *K* replicas)	Fixed # of replicas for each GUID (each GUID has *K*+1 replicas)
Caching	No caching, tries to achieve load balancing by adjusting number of replicas	Caches GUID→Network Address Mappings to increase availability of mappings for "hotspot GUIDs"	Future work

Table 1. Comparison Table of Auspice, GMap and DMap.

We summarize this chapter's contents in Table 1. Having established some of the key components and differences in each of the Global Name Resolution System designs, it is necessary to consider the key resources involved in each GNRS and each of their vulnerabilities.

4. Key Resources and Possible Vulnerabilities

This chapter identifies key resources in each of the GNRS system designs and forms a preliminary analysis from reading the DMap publication [9], Auspice and GMap technical reports [12, 15], and discussing the GNRSs' design. We first describe how an attacker can use a Rainbow Table to target specific routers in the system, and review a key insight gained by examining a Rainbow Table generated for DMap. We follow this with a discussion of three basic approaches an attacker can take to disrupt service in a GNRS system: volume based attacks, attacks that corrupt data, and attacks that involve a malicious Autonomous System. Through all of these approaches, we consider how they affect the key resources in a distributed network system: computation power, bandwidth and storage.

4.1 Targeting with a Rainbow Table

An attacker's ability to target a specific server can affect all of that server's basic resources: its network bandwidth, its computation power and its storage. In order to assign GUID mappings to servers, DMap and GMap use a well-known hash function that, given a GUID, outputs a network address to store the mapping on. The hash function being static and well known gives an attacker an advantage when targeting GNRS servers, because the attacker can invert the hash function using a Rainbow Table[3]. Such a table can be created by mapping all GUIDs to the servers their mapping is stored on and keeping a chart of which GUIDs are stored on which servers.

Then, an attacker can reverse the hash function, that is, look up which GUIDs will be on a specific server, instead of which server hosts a GUID. Once they have a list of GUIDs that are on

[3] Rainbow Tables are commonly used to invert cryptographic hash functions, see [32].

a server they want to attack, they can send messages to all of these GUIDs flooding their target server with traffic. Initially, if messages are sent fast enough, this is an attack on the bandwidth to the targeted router, and if requests do get through, the computation power of the router may be strained. Either way, this attack should deny the resolution for most, if not all, GUIDs on the router. If attack messages are sent slowly so that they are successfully inserted, an attacker could attempt to send incorrect data, corrupting the storage of the target server.

Auspice mitigates a similar type of attack by using epochs and necessitating system involvement that could easily include gatekeeping. In Auspice, a host (or a server on its behalf) has to ask replica-controllers where a GUID-NA mapping is for new GUIDs. This makes filtering possible at the server and replica-controller levels; they could notice if a specific host is asking for the active replicas for an unusual number of GUIDs. With both rate-limiting/filtering and a shuffle every epoch, Auspice makes building up an accurate Rainbow Table for which active replicas store which GUIDs rather challenging. In section 4.2, an attacker will use a Rainbow Table to determine desired GUID(s) that should be stored on a certain server (or nowhere at all).

4.1.1 System Insight from a Generated Rainbow Table

One can also gain insight as to how a system functions by examining a Rainbow Table. Generating a Rainbow Table demonstrated that DMap does not always map to *K* distinct Autonomous Systems. In an experiment, we ran an instance of DMap modified to generate a Rainbow Table. We set *K* equal to three and equally distributed the IPv4 address space among 128 ASs, simulating each AS announcing 33,554,432 addresses. Since DMap runs *K* independent hash functions, it should determine *K* distinct ASs to hold mappings for each GUID. This DMap instance mapped 10,000 sequential GUIDs, 224 of which mapped to less than three ASs, about two percent. This means that DMap does not check for unique ASs when determining the mapping, allowing one of the *K* mapping placements to collapse into another. This decreases the system's resilience to failure. In another experiment, with the same parameters for autonomous system prefixes and *K*, a Rainbow Table was generated for 1000 **random** GUIDs, 29 of which mapped to only two ASs. Each AS has a limited number of gateway routers through which all external traffic must flow. For popular GUIDs, the *K*-1 ASs that do have a mapping must handle all traffic looking for the specific GUID-NA mapping.

There are a couple different ways to handle this idiosyncrasy. First, there could be a designated "backup" hash function that is run in this situation. Instead, this could be addressed by treating the duplicate Autonomous System as an "IP hole" and following the existing IP hole protocol. Finally, one could simply increase *K* such that it is irrelevant if a GUID's mapping is only stored on *K*-1 servers. GMap reduces the likelihood of such AS collisions in yet another way. GMap explicitly ensures that at least three (one global, one local and one regional) replicas are on different servers. The only possible server-collisions are among the replicas at each geographic level.

4.2 Volume Based Attacks

In this section, we review attacks based on high traffic volume that flood the servers of a GNRS with requests or other traffic. We consider flooding attacks in Auspice, GMap, and then DMap, pointing out different strategies an attacker would use in each system. The key resources in Auspice are the active replicas and the replica-controllers. They need to store a lot of information and be available for clients to look up the location of GUIDs. Although active replicas are intended to handle frequent and voluminous requests, directing a similar request load to replica-controllers is not expected in the system design: “In practice, we expect replica-controllers to be contacted infrequently as the set of active replicas can be cached and reused” [11, p. 6]. The Auspice design assumes limited contact with replica-controllers, but does not enforce limited contact. If an entire botnet was directed to request new names at the same time it could most likely overwhelm the replica-controllers managing those names, making replica-controllers vulnerable to such volume-based attacks.

Similarly, an entire botnet simultaneously moving (or claiming to move) from one set of Network Addresses to another can overload the active replicas, stressing their computation al capabilities. Whenever a GUID moves from one Network Address to a Network Address (attached to another Autonomous System), its name record must be updated to reflect the change. Since the active replicas have a Paxos instance for each GUID, the update will be coordinated and slower than if each active replica was performing it individually. A malicious Local Name Server could take advantage of having a list of all active replicas for a GUID and forge or replay different update messages to each of the GUID’s active rep licas, forcing them to resolve closely timed but conflicting updates.

As with Auspice, GMap is also vulnerable to a lookup flooding attack. The way GMap looks up GUIDs makes its processing power particularly vulnerable to a lookup flooding attack. When a local GNRS-enabled router receives a lookup request, it checks its cache for that GUID. For each cache miss, the local router must then compute all replicas for the requested GUID. An attacker can require a local GNRS router to do a lot of wasteful computation if they send many lookup requests for far away, unpopular GUIDs at once. Using unpopular GUIDs is important so that they are not in the local server's cache. Using GUIDs known to be far away is what makes the server perform wasteful computations. What makes GMap particularly vulnerable to this kind of attack is its assumption that a majority of requests for any given GUID come from devices geographically co-located with that GUID[4]. When the GUID being looked up is not located near the GMap server handling the query, the server handling the query must compute three different sets of hashes for the GUID. It runs local, regional, and global hash functions on the GUID, querying servers in each category in turn, as was discussed in Chapter 3 and demonstrated in Figure 6. Consider an attacker with access to device Y who wants to attack Y's local GNRS router, R_Y. The attacker requests resolution of GUIDs he knows to be located far away from Y (or nowhere at all), requiring R_Y to send three times as much traffic as the attacker (local, regional, then global), negatively impacting its available bandwidth for legitimate requests. This attack could also affect R_Y's storage, since it must maintain state for the servers it is waiting on.

Despite this vulnerability, there are two mitigating factors that work in GMap's favor against this attack. First, the attacker must use a botnet if he desires to attack a target not in his geographic region. Secondly, this attack only targets a single router, which is arguably not a core part of GMap. Routers fail all of the time, and traffic gets redirected to other routers. While this attack could be very successful on a specific router, it will only deny service when other routers are not available.

[4] GMap is evaluated for three query localities [15]:
"strong query locality": 60% of GUIDs are local, 20% are regional and 20% are global
"medium query locality": 20% of GUIDs are local, 60% are regional and 20% are global
"low query locality": 20% of GUIDs are local, 20% are regional and 60% are global

DMap is open to a similar attack on its computation power in which an attacker requests GUIDs that hash to an IP hole. The server trying to resolve the lookup has to rehash the GUID multiple times, per the IP hole procedure, whenever a request for them comes in. After each hash, the server must consult its routing table in order to ensure that the GUID cannot be stored at an Autonomous System serving the address corresponding to the hash. Since the IP hole protocol is not implemented in the current version of the DMap code, a similar but less sophisticated DMap attack is demonstrated and discussed in Chapter 5.

4.3 Corrupting Data

Instead of sending a large quantity of data to parts of the GNRS system, an attacker could focus on sending – or making the servers interpret – false or misleading data. We examine this problem in both Auspice and GMap.

4.3.1 Corrupting Data in Auspice

Data corruption often provides a subtle approach to executing denial of service attacks. The majority of this section is dedicated to a consideration of different ways to interfere with Auspice's demand aware placement through corruption of data. We also discuss how MobilityFirst's GUIDs certify themselves, and discuss attacks based on how Auspice updates its data and the seemingly limitless amount of data Auspice offers to store.

Tricking the geo-distribution is an attack that causes more bandwidth to be used per request, increasing latency and response times. As was discussed in Chapter 3.2.2, Auspice's demand aware placement considers both the location and volume of requests for a GUID's name record and the frequency of updates to a GUID's name record when determining where to place active replicas. An attacker can attempt to interfere with the statistics kept on one or both of these metrics.

One approach to creating this interference is to bias the replica-controllers' demand estimate, making them estimate more demand for a name or subset of names in a region far away from where peak legitimate demand is, in order to worsen or deny service to the legitimate demand. This could be accomplished by employing a botnet in a region geographically distant from legitimate peak demand to send many requests for the target name or names to change the geo-distribution of demand for those names. This will cause the replica-controllers to place a portion of active replicas for target names that should be near legitimate demand near the botnet instead. Auspice does limit the efficacy of this attack by updating active replicas each epoch, but a motivated attacker can persist in sending the botnet's requests.

Another way to exploit the distribution, which takes the update rate into account when determining the number of active replicas, is to update a name very frequently such that the replica-controllers limit the number of active replicas according to the Demand-Aware Replica Placement equation discussed in Section 3.2.2. This assumes that an attacker can spoof the target GUID, which is supposed to be self-certifying. The feasibility of self-certifying GUIDs is questionable. The authentication procedure as suggested by MobilityFirst [5] is shown in Figure 8. With this authentication scheme, if GUID Y's private key is every exposed, the device that used GUID Y as its identifier would need to obtain a new GUID. A separate issue from Y's private key becoming compromised is the size of the nonce, *n*. A GUID should be authenticated whenever it starts a new session or connection, and if *n* is reused, the values of K^+ and $K^-(n)$ could just be replayed. One solution would be to first encrypt *n* so that devices listening on the network cannot tell if *n* is being reused. Establishing a symmetric key between X and Y to encrypt *n* would incur further overhead. It is possible that initiating the authentication scheme becomes a way to attack devices or Network Addresses (which are also self-certifying GUIDs).

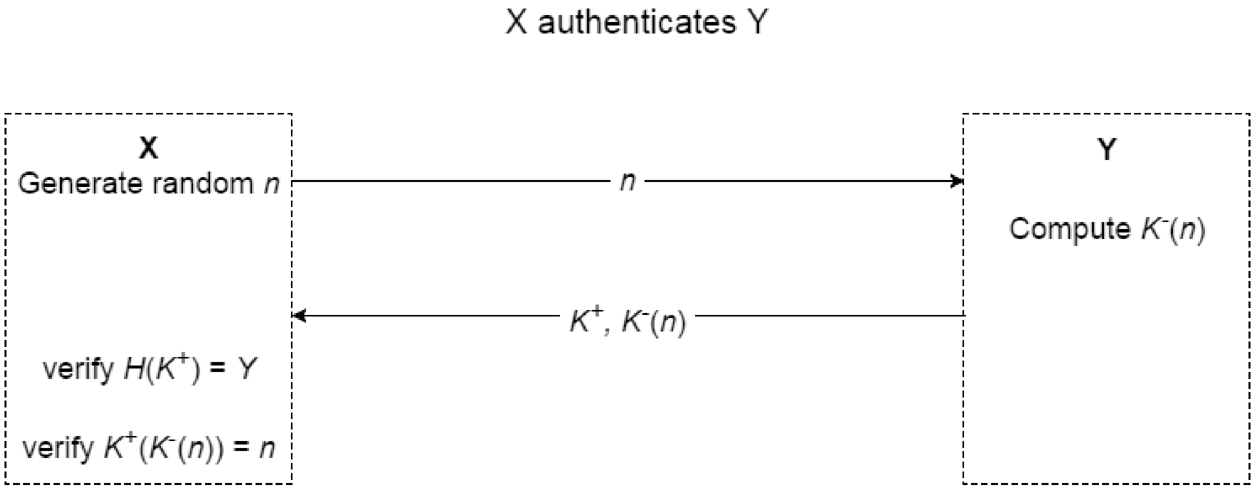

Figure 8. ***Self-certifying GUID.*** *GUID Y has a private key K^-, public key K^+, such that a well-known one-way hash function, when applied to the public key, is equal to the value of GUID Y.*

If self-certifying GUIDs are not resistant to impersonation, DMap is susceptible to a similar attack where inserts are created or modified in flight to have very small timeout values, which are sent by the inserter.

A different way to corrupt data in Auspice is to take advantage of the Auspice system enforcing total write ordering for updating GUIDs (especially lists). An attack of this type requires a more detailed analysis of the custom version of Paxos used in Auspice, Gigapaxos, for any additional attack surface it brings to the system design.

Finally, in Auspice, an attacker could just try to fill up the database with arbitrary (key, value) pairs that serve no purpose other than to occupy disk space. Auspice's "super columns", mentioned in [12]'s Section 3.2.5, are multi-purpose, customizable and extensively infinite, with no described or suggested limitations other than an access control list.

Vu et. al. address similar concerns in DMap by defining name records as simple GUID-NA mappings and claiming that each GUID can only be associated (multi-homed) with up to 5 NAs [9], but this is not currently enforced in the implementation available at [10].

4.3.2 Corrupting Data in GMap

Another means of attacking a GNRS's storage resources is possible in GMap. An attacker can try to limit the availability of GMap's cached "hotspot GUIDs". This is similar to interfering with Auspice's demand aware placement – both attacks involve misrepresenting legitimate use of the GNRS in order to direct GNRS resources away from where they would be most useful. This is a tradeoff with tailoring a system to treat GUIDs according to how frequently they are used. Filling up the cache with nonsense and preventing the cache from caching legitimate hotspot GUIDs is especially easy since the cache replacement scheme is Least Recently Used (LRU) and not Least Frequently Used (LFU). Caches have limited storage space, and a LRU cache will make space for new entries by discarding the entry that was requested longest ago. A LFU cache keeps track of how often cache entries are requested and makes space by discarding entries that are requested infrequently. GMap's goal in caching is specifically to make hotspot GUIDs, defined as GUIDs that are most frequently requested, available. When using a LRU cache, it is easy for an attacker to make the cache discard these popular GUIDs by rapidly looking up different unpopular GUIDs. The attacker can run a loop over a number of GUIDs approximately equal to the size of the cache and keep requesting them. The attacker only needs to look up each GUID once (albeit recently) for it to take precedence over a more frequently looked up

GUID. If the cache were Least Frequently Used, an attacker would have to request the same GUID repeatedly until its request rate surpasses the least frequently requested GUID in the cache before having an effect. This attack targets system storage. However, since the caching is intended to decrease bandwidth usage, with a successful attack of this nature, one would expect bandwidth usage to increase.

Instead of making servers store irrelevant, unpopular GUID records, an attacker could forge an acknowledgement with an incorrect NA for a target popular GUID. This attack is also a blow to storage and bandwidth. It doesn’t overflow the storage, but causes the system to store corrupt data. Since the client does not receive a NA that actually has the GUID, the client will likely submit the request again, using more bandwidth.

DMap does not have caching implemented currently and leaves it as future work. If its caching is similar to its successor, GMap, it will be vulnerable to similar attacks. DMap does not verify that GUIDs sending insert requests are actually located where they claim to be. If DMap authenticates GUIDs, as reviewed in 4.3.1, then each GUID could only poison its own name record. The consequences of verifying a GUID’s presence at its Network Address(es) is discussed in Chapter 6, and other attacks against DMap system storage are discussed in the next section.

4.3 Malicious Autonomous System

The previous sections of this chapter have focused on what an attacker can do with a botnet or other limited resources. We now consider the ramifications of an attacker controlling an entire Autonomous system, or at least an Autonomous System’s Border Gateway Router. Attacks in this section initially target storage within a malicious or other Autonomous System, and, when successful, affect the bandwidth usage on links to the other copies of name replicas.

A malicious Autonomous System (AS) that announces and is therefore assigned a specific range of the name space, can simply not cooperate by storing DMap mappings of GUIDs to NAs. This would require an attacker to control a significant portion of the name space in order to receive a significant portion of the mappings. One way around that would be to create a malicious AS that claims to host the addresses in the IP hole and becomes black hole for GUIDs that hash to the IP hole(s). When successful, this causes the other servers hosting mappings for affected GUIDs to experience more requests.

A malicious AS could also forge GUID migration messages to ASs holding legitimate mappings telling them that they are Deputy ASs and to drop their mapping, disrupting the storage of name records. If the attacking AS manages to force all *K* replicas of the mapping to drop, the GUID will be unresolvable, completely denying service. If only some of the replicas are dropped, the servers hosting the other replicas may experience congestion. In order for legitimate migration messages to work when the Deputy AS is actually a deputy, upon receipt of a migration message, the Deputy AS would have to either automatically drop the mapping, or check if it is rightful owner of the mapping before dropping it. This computation-intensive checking could be mitigated by including a Boolean value with the mappings on each server, indicating whether the server is acting as a deputy for that record. Keeping an additional true/false value would require minimal additional storage, but would allow the computation to only be performed once, when a GUID is inserted.

Auspice limits the impact of a rogue AS or group of servers by using replica-controllers for oversight and breaking up time into epochs. If a group of servers just refused to resolve GUIDs to NAs, the replica-controllers could remove active replicas placed there. A malicious server could try to report false statistics and claim that a lot of requests for a specific GUID came to it and were resolved quickly.

4.4 Summary

In this chapter, we began by discussing an important tool in a distributed system attacker's toolkit, the Rainbow Table. We followed by considering three basic approaches an attacker could use to deny service to all or part of a Global Name Resolution Service: sending large amounts of data, sending corrupting data, and taking over a system resource. Unsurprisingly, significant attack opportunities arise where the systems address edge cases. These include the IP hole problem in DMap, the consistency guarantee provided by Auspice provided by Paxos, and GMap's unique consideration of locality-based inquiries. We provide a quick overview of the keep points in this chapter in the following table.

Basic Resource Targeted	Auspice	GMap	DMap
Computation	Large-Scale Simultaneous Mobility overloads Active Replicas, which use Paxos	Large-Scale Simultaneous Mobility – have to remove mapping from old local and regional servers and add to new local and regional servers	Large-Scale Simultaneous Mobility would cause inconsistent mappings for same GUIDs
	Large-Scale Name Lookup overloads Replica-controllers	*Individual routers hash GUIDs to know which replicas to ask*	*Individual routers hash GUIDs to know which replicas to ask*
	Send Insert Requests (with many, many entries) for "fake" GUIDs		
Bandwidth	Interfere with Demand Estimate	Request far-away GUIDs to waste local/regional requests	
	Interfere with Update Frequency Estimate		

	More central control, easier to filter. Attacker alone can’t determine which GUIDs are where	Flood all GUIDs on a specific router using Rainbow Table Is this worse than just asking that router for GUIDs it doesn’t actually have?	
Storage	Paxos/Total Write ordering: effect on active replicas **and** replica-controllers		
		Cache Overflow	
		Forge ACK with incorrect NA for target GUID (possibly different than impersonating GUID depending on structure of ACKs)	
	Impersonate GUID, store wrong NA		
	Insert GUID multihomed at 1000 addresses		

Table 2. ***Summary comparison of possible vulnerabilities affecting key resources.*** *Italics symbolize a design choice that is not tied to a significant vulnerability.*

5. DMap Server Attack

In this chapter, we look at the Internet equivalent of someone dumping a truckload of fake "change of address" forms at your local post office right before you submitted your change of address postcard. This type of attack is present in today's Internet and often referred to as a DNS flooding attack. The possibility and effectiveness of a GNRS flooding denial of service attack on DMap was demonstrated in a testbed. As discussed in Chapter 3.1, DMap is a direct mapping implementation of a GNRS that uses *K* well known, independent hash functions to determine where a name record is stored for each GUID. A volume based attack was conducted where a target DMap server's socket buffer gets filled with requests from an attacker, so that legitimate client requests take longer or are just dropped. A Network Address, the endpoint of my experiment, maps to our concept of an AS, so I am not concerned with intra-AS topology. Malicious behavior representing attacks was added. Our analysis, as presents in Section 5.4, leads us to conclude that approaches to mitigating this sort of flooding DoS attack will, in turn have a negative impact on supporting mobility, a central objective of the MobilityFirst project. This is an example of the tradeoff design challenges exposed through DoS opportunity analysis.

5.1 Recursive Inserts and Updates

As shown in Figure 9, Insert messages coming from end-users running the client version of DMap protocol set a recursive option to True so that the Insert is replicated *K* times. Then, the local DMap server runs the *K* hash functions, stores the GUID's network address if necessary (because it is one of the *K* servers, for local replication or due to caching), and then forwards a non-recursive version of the request to the remaining servers determined by the hash functions. After sending the remote requests to other servers, the original server, 45.0.0.5 in the figure, keeps information on G_X's entries in an "awaiting responses" map. Note that the other servers (128.5.5.5 and 192.5.6.7) do not forward the request to each other, since they have a non-recursive version. After 128.5.5.5 and 192.5.6.7 run the hash functions and insert G_X's network address in their storage, they send Insert Response messages back to 45.0.0.5, which, in turn, sends a message back to G_X that its insert was successful.

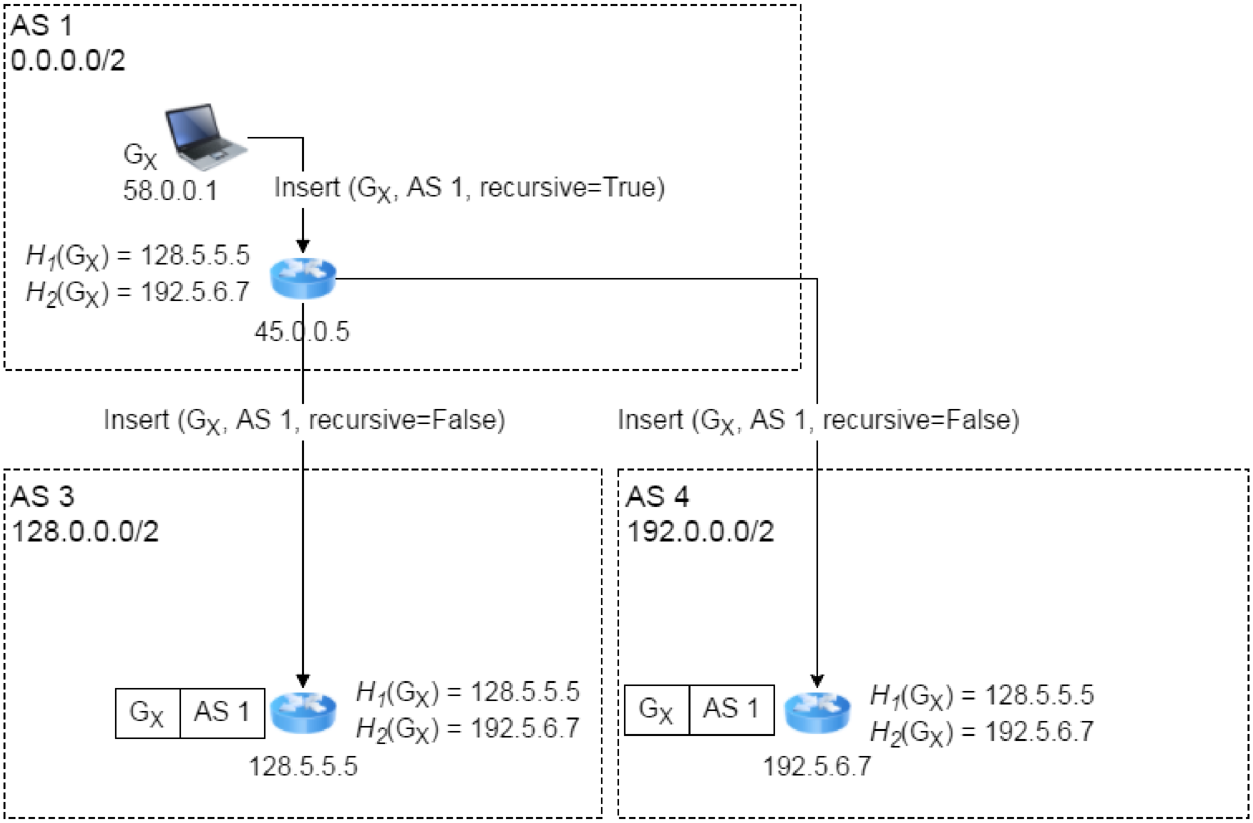

Figure 9. DMap Normal Operation with K=2. *Setting K=2 means that G_X's network address is stored on servers corresponding to two independent hash functions. So that the first DMap server knows it must pass the Insert request along to other DMap servers, the request is marked as "recursive" by G_X, the device sending its Insert message to 45.0.0.5. Server 45.0.0.5 runs both hash functions, removes the recursive option from the Insert message and forwards it to the servers corresponding to the hashes of G_X.*

5.2 Attack Setup

Before setting up the attack, we had to determine an appropriate environment in which to run the experiment, obtain the DMap source code, and set up DMap servers, including a modified attack server. The modified attack server, along with the configuration files to run the experiments, are available in [17].

5.2.1 ORBIT Testbed

We chose the Open-Access Research Testbed (ORBIT), which was designed for Next-Generation Wireless Networks and runs on the Internet [18], as our experimental environment. Since it uses the same links that support the Internet, link behavior is realistic but not customizable. Simulations were used to evaluate Auspice, DMap, and GMap, but each simulation focused on different parts of the architecture that the system designers wanted to evaluate, leaving the simulations incomplete in different ways. Since we wanted to compare the GNRSs, running them all in the same testbed was a planned control in the experiment. ORBIT uses the ORBIT Management Framework [19] to execute experiments. Outages on the main grid led us to run the experiment on a more reliable ORBIT testbed, but its much smaller scale limited what we could do. In the experimental setup, most ORBIT nodes act as an abstraction for an Autonomous System. We are not concerned with intra-AS traffic because it is independent of the GNRS. Once traffic gets to an AS, the AS can route it to the specific device in whichever way it sees fit. DMap is implemented in ORBIT, but not fully. Their solution for the IP hole problem mentioned in section II.B of [9] is not implemented in the DMap code obtained from [10], so the AS mapping used in the experiment covers all IPv4 addresses. Additionally, updates are not included in the client, and the migration messages are not included in this version of DMap, so we focused our experiments on inserts and updates.

5.2.2 DMap Server Modification

First, let us consider the feasibility of an attacker successfully changing DMap server source code. DMap servers are co-located with Border Gateway Protocol (BGP) servers. Since the

DMap “hashing, rehashing and prefix matching processes are done locally by the border gateway” [9, p. 4], the DMap servers are not logically separate from BGP servers. Therefore, a reasonable assumption is that the DMap servers would have similar security to BGP servers. Since BGP is still functionally insecure [20], it would be feasible for an attacker to modify code on a DMap server, especially since they can use any server as the attacking server. The server code on the attack server in the experiment was changed to generate Insert requests, which usually come from a device running the DMap client. These requests were spawned according to a variable *attack frequency*. Unlike the client, the attack server did not waste its memory keeping state on these attack Insert messages. Unlike a behaving GNRS Server, the attacking server did not remove the “recursive” setting from these generated requests. Finally, instead of hashing the GUID in these generated requests to determine which servers the request would be sent to, the list of all DMap servers was used.

This modified server still processed external requests normally in order to not raise suspicion. Recall from Chapter 4.1 that the *K* hash functions are well known so that any DMap server that receives a Lookup request can determine which server(s) to ask for the name record of the particular GUID. This avoids a bottleneck at a central resource such as Auspice’s replica-controllers. However, the tradeoff is that it is easy to generate a list of all GUIDs that hash to a target server. A Rainbow Table was generated and used to pick a GUID that corresponded to the target DMap server. The GUID was the same for all attacker-generated Insert requests to demonstrate the proof of concept of this attack.

In order to analyze the effectiveness of the attack, tcpdump, a Unix command that can print to file the contents of packets on a network interface, was run on all DMap servers participating in the experiment. Although many attackers do not measure the effectiveness of their attacks, the ability to do so may help them remain attack effectively and remain undetected. We now outline how they could do so without special access to the target server. Since the target DMap server must accept Insert requests from arbitrary IP addresses (see Section 5.4 Tradeoffs), including that of the attacking server, it is safe for the attacker to assume that the attacker-generated insert requests are succeeding with a similar rate experienced by legitimate clients. Under this assumption, the attacker could vary the attack frequency and use different GUIDs that all mapped to the target server (from the Rainbow Table), and keep track of which inserts succeed to measure the effectiveness of differing attack rates.

If DMap were deployed, many different clients would be inserting themselves into DMap and looking up the locations of other devices. Since we had a limited number of nodes on ORBIT, one node acted as multiple clients, sending multiple insert requests to th e target server. Funneling all client traffic through the same node also allowed us to easily measure the client's success rate without varying link speeds and topologies between the client and target server. As shown in Figure 10, the client sends 1,000 inserts for GUIDs 1 to 1,000 in sequence. The DMap client takes in a parameter, *t*, and sets a 1000**t* nanosecond = .01**t* microsecond delay between client messages. As shown in Table 3, an efficient client delay was determined to be *t*=4, yielding a throughput of 1000 inserts per second.

t	Throughput
2	967 inserts per second
4	1000 inserts per second
16	821 inserts per second

Table 3. Client Insert Throughput

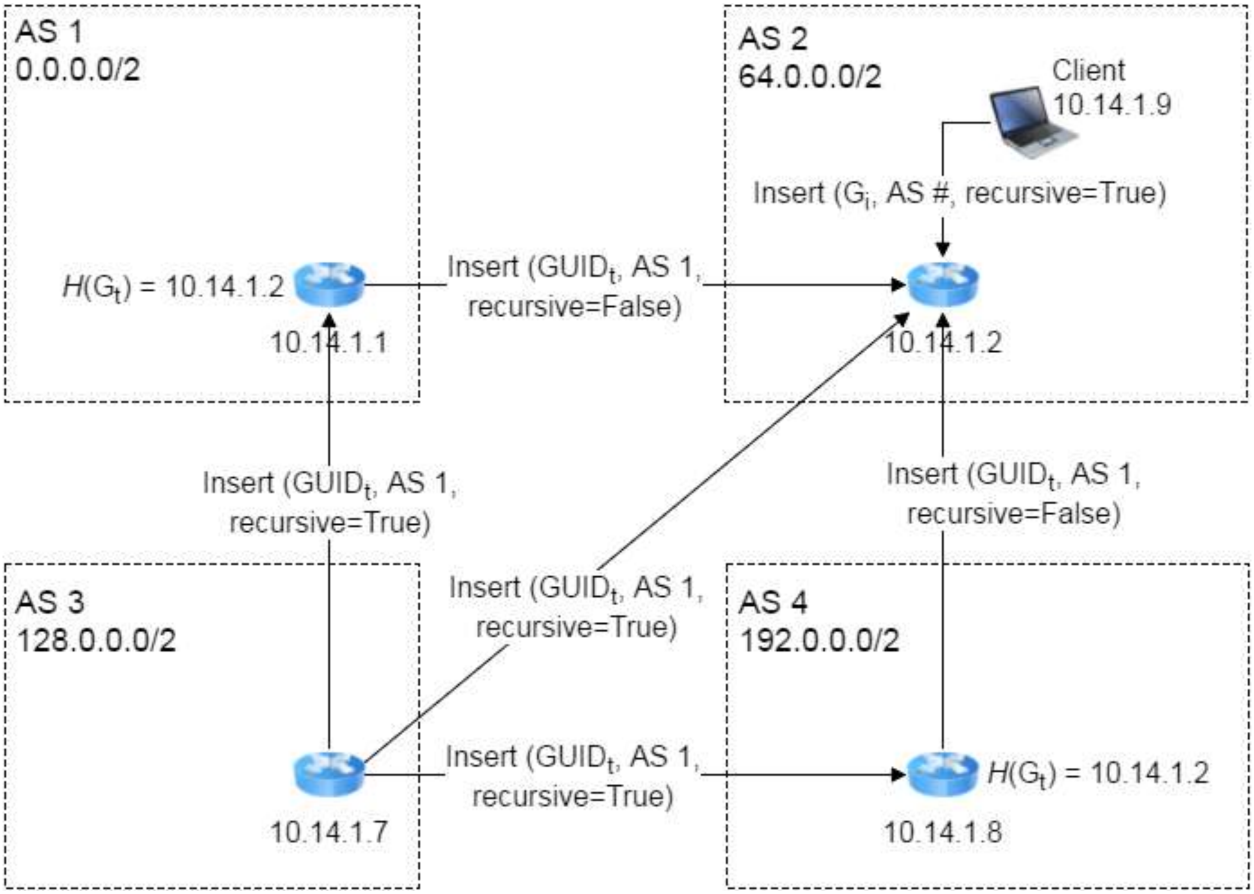
AS 1
0.0.0.0/2
H(Gt) = 10.14.1.2
10.14.1.1
AS 2
64.0.0.0/2
Client
10.14.1.9
Insert (Gi, AS #, recursive=True)
10.14.1.2
Insert (GUIDt, AS 1, recursive=False)
Insert (GUIDt, AS 1, recursive=True)
Insert (GUIDt, AS 1, recursive=True)
Insert (GUIDt, AS 1, recursive=False)
AS 3
128.0.0.0/2
10.14.1.7
Insert (GUIDt, AS 1, recursive=True)
AS 4
192.0.0.0/2
H(Gt) = 10.14.1.2
10.14.1.8

Figure 10. Our Attack*. The client sends an Insert Request every 4 µs, one for each of G_i ϵ [1, 1000] to the target DMap server 10.14.1.2. The modified DMap server, or attack server, 10.14.1.7, sends recursive Insert Requests to all of the DMap servers with a GUID known to map to the target server.*

5.3 Results

As explained in the previous section, a DNS flooding attack was run in the ORBIT testbed. DMap has built-in functionality to generate Cumulative Distributions of some statistics. The "delay" mentioned in the graphs is the amount of time between when the first attacker packet arrives at the target server (10.14.1.2) and when the first client packet arrives at the target server. A negative value, as in Figure 11, indicates that the client's packet arrived at the target server before any of the attacker's packets. Timestamps were determined by inspecting a tcpdump running on the target server during each experiment. Figure 11 demonstrates that, as the attacker sends inserts at a faster rate, the insert round-trip times experienced by the client increases. One can observe some variation even among the two baseline trials without the attacker. When the attacker was sending inserts every 4 ms, 1,000 times slower than the client, only 604 of the client's 1,000 inserts were successful. For the trials in Figure 11, the attacker's first insert arrived at the target less than a second before the first client's insert.

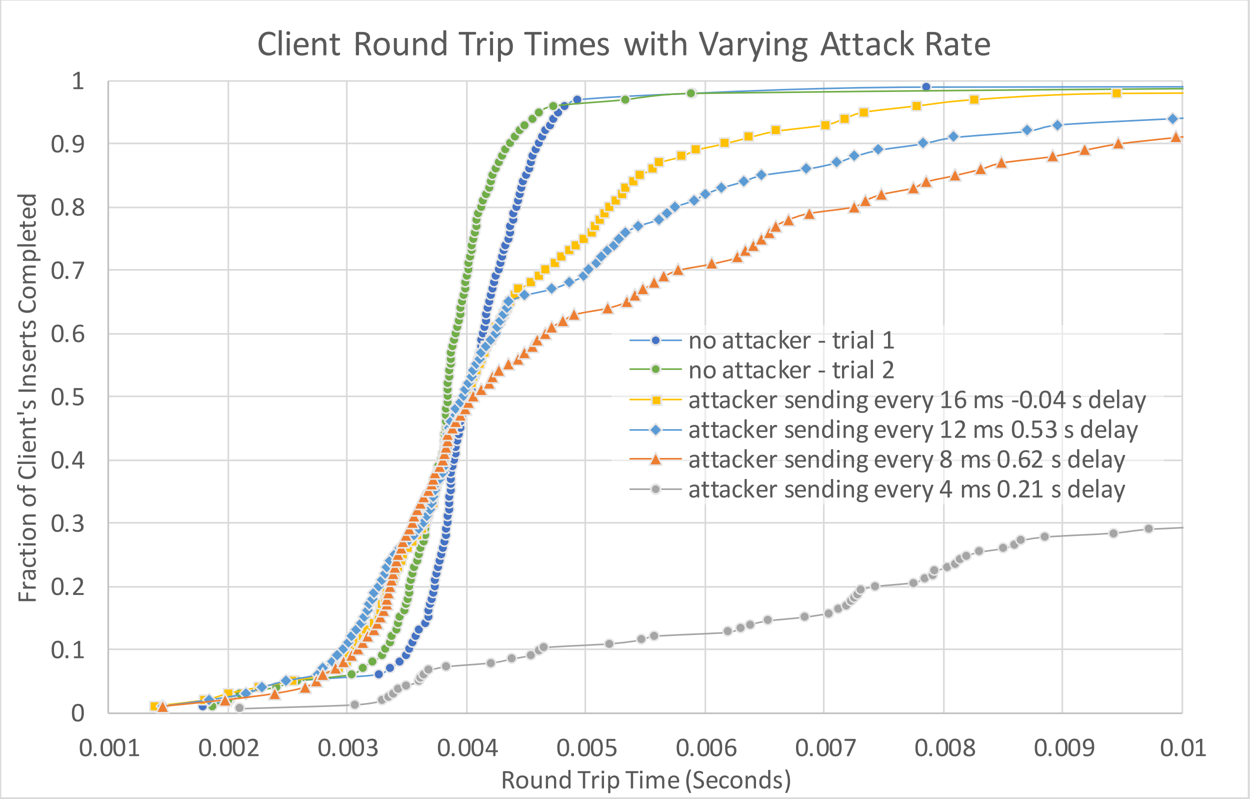

Figure 11. Cumulative Distribution Function of Client Round Trip Times with Varying Attack Rate. These trials were run with the purpose of minimizing the delay so that the effect of the attack rate could be observed. When sending at 16ms, the attacker's first packet arrived at the target server after the client's first packet arrived, despite starting the attacker before the client.

Figure 12 shows what happens when the attacker has more of a head start. In order to get a sense of scale between Figure 10 and Figure 11, observe that the series "attacker sending every 8 ms 0.62 s delay" appears in orange triangles in both graphs. This suggests that the amount of time the attack runs is a stronger indicator of a successful attack than the rate at which the attacker sends packets. This is further supported by the fact that for some longer delays, the client was completely blocked from inserting any GUIDs. A trial with the attacker sending every 8ms was performed with a 32.70 second delay, but none of the client's inserts were successful. Another trial with the attacker sending every 4ms with an 11.07s delay also prevented the client from having any successful inserts.

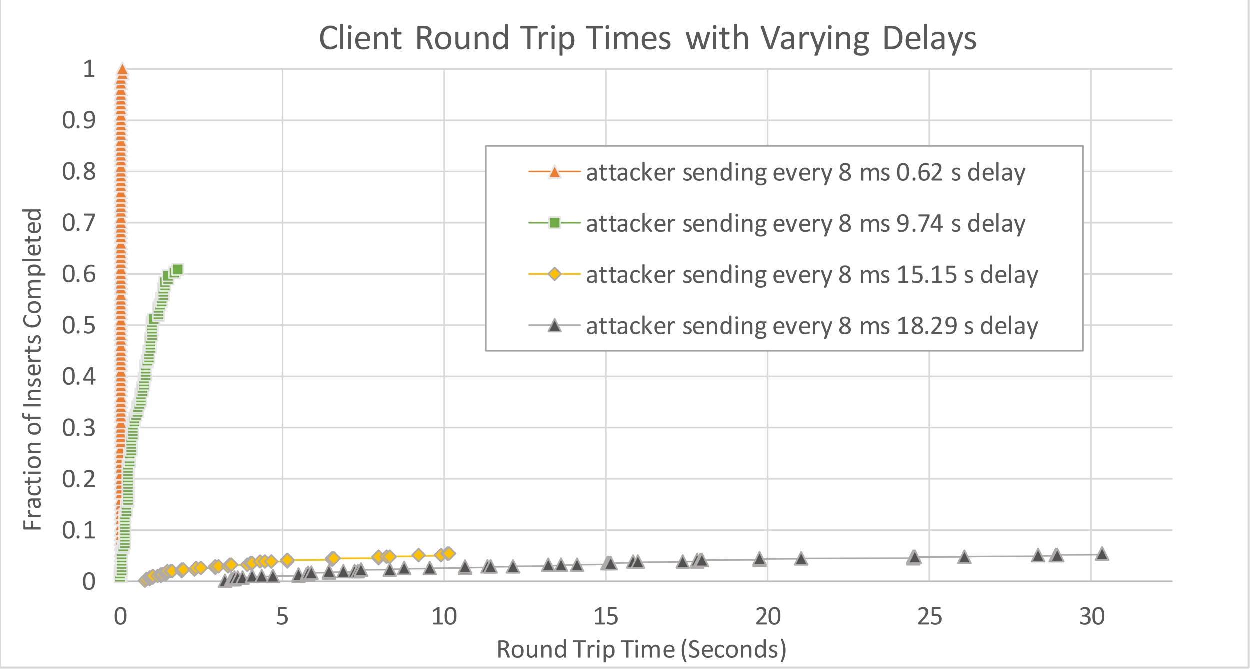

Figure 12. Cumulative Distribution Function of Client Round Trip Times with Varying Delays. This graph shows trials emphasizing the delay between the arrival of the first attacker's packet at the target server and the arrival of the first client's packet at the target server. Only 604 insert requests ever get completed by the client for the 9.74s delay, and only 53 are completed for each of the 15.15s and 18.29s delays.

Next, the DMap statistics kept by the target server were analyzed.

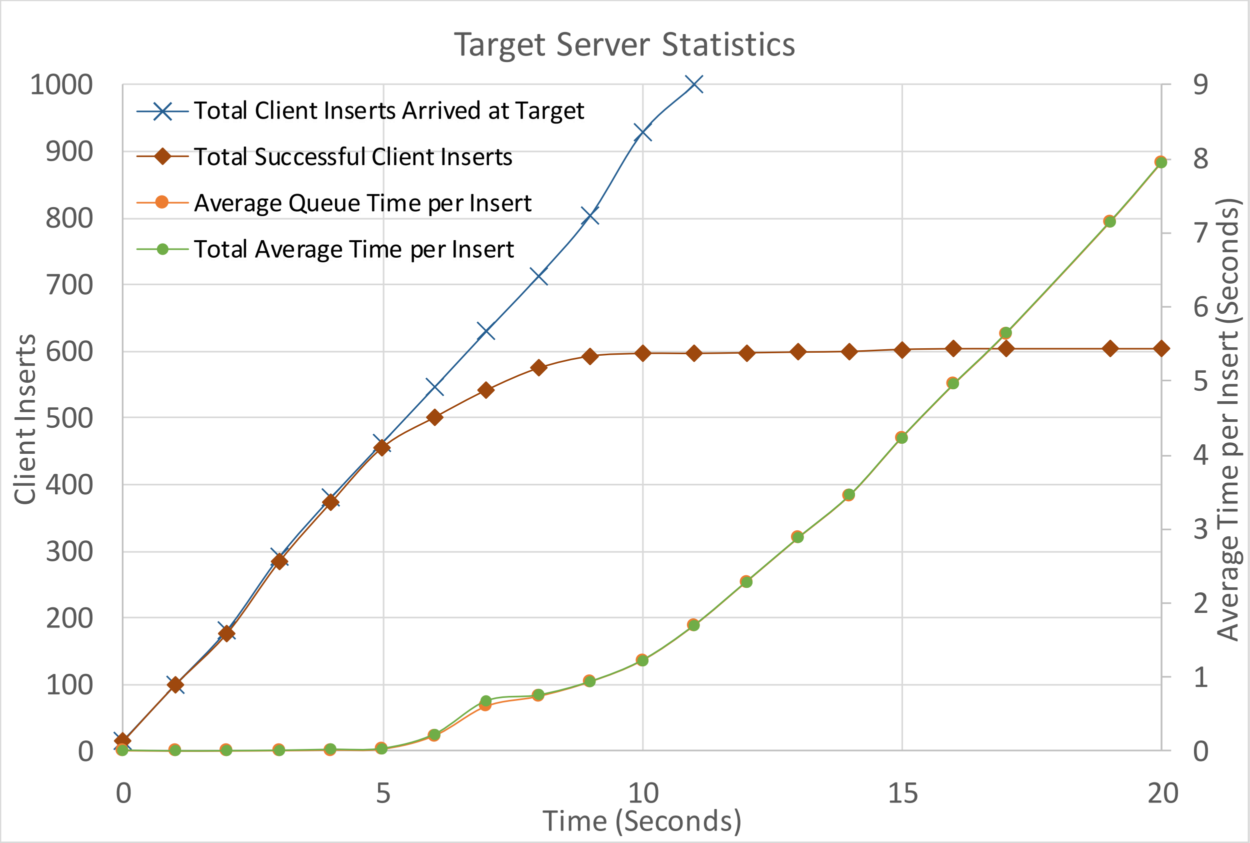

Figure 13. Target server statistics for the "attacker sending every 4 ms 0.21 s delay" trial.

As seen in Figure 13, the orange and green trend lines are barely distinguishable. We conclude from that factthat the Total Average Time per Insert is almost entirely made up of the Average Queue Time per Insert. The Total Client Inserts Arrived and Total Successful Client Inserts were determined from a tcpdump trace running on the target server. Packets sent from the client to the target were inserts, and each response back to the client from the target server is an insert response, acknowledging a successful insert. For the first five seconds, client inserts are completed at about the same rate they arrive at the target server. Around 5 seconds into the experiment, the average queue time per insert starts to increase, and continues increasing. At the same time, a noticeably smaller fraction of client inserts are successful, until 10 seconds when they grind to a halt – between 10 and 20 seconds, only 7 client inserts were successful.

Recall from Figure 9 that when a client inserts something into DMap, its first point of contact, the target server in our experiment, needs to forward the inserts to the appropriate servers to store the information contained in the insert. In order to successfully complete an insert, the target server needs to receive responses from those remote servers that store the contents of the insert. If the target server's socket buffer is full, not only can it not receive inserts from the client, but it can also miss responses from these remote servers. This phenomenon is supported by Figure 14, which shows the number of outstanding responses increase after 5 seconds and a huge spike in average remote time per insert.

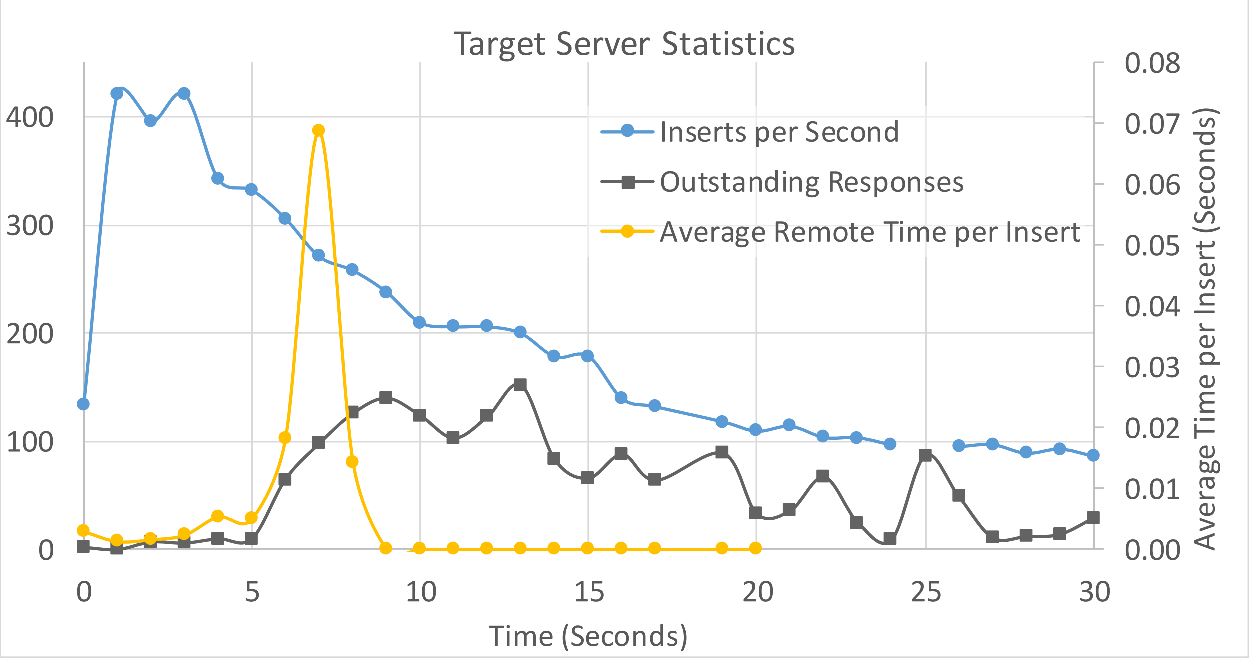

Figure 14. Inserts per Second, Outstanding Responses and Average Remote Time per Insert, measured by the Target Server.

Finally, we examined the effect of the malicious DMap server on client Lookups. The attacking server sent attacks at a slow rate to ensure that all of the GUIDs being looked up were eventually able to be successfully inserted by the client. After the 1,000 GUIDs were inserted, lookup requests were sent for each of them. One might think that lookup would be less affected by the DNS flooding attack, since only one of the *K* servers with a GUID's mapping on it must be contacted. However, the client's only point of contact in DMap is its closest DMap router, which has been flooded with inserts from the attacker. This increases how long it takes lookup requests to get through the target server, as shown in Figure 15. Without the attacker, 95% of lookups were completed in 3.94281 ms, but with the attacking DMap server it took 75.46862 ms, over 19 times slower.

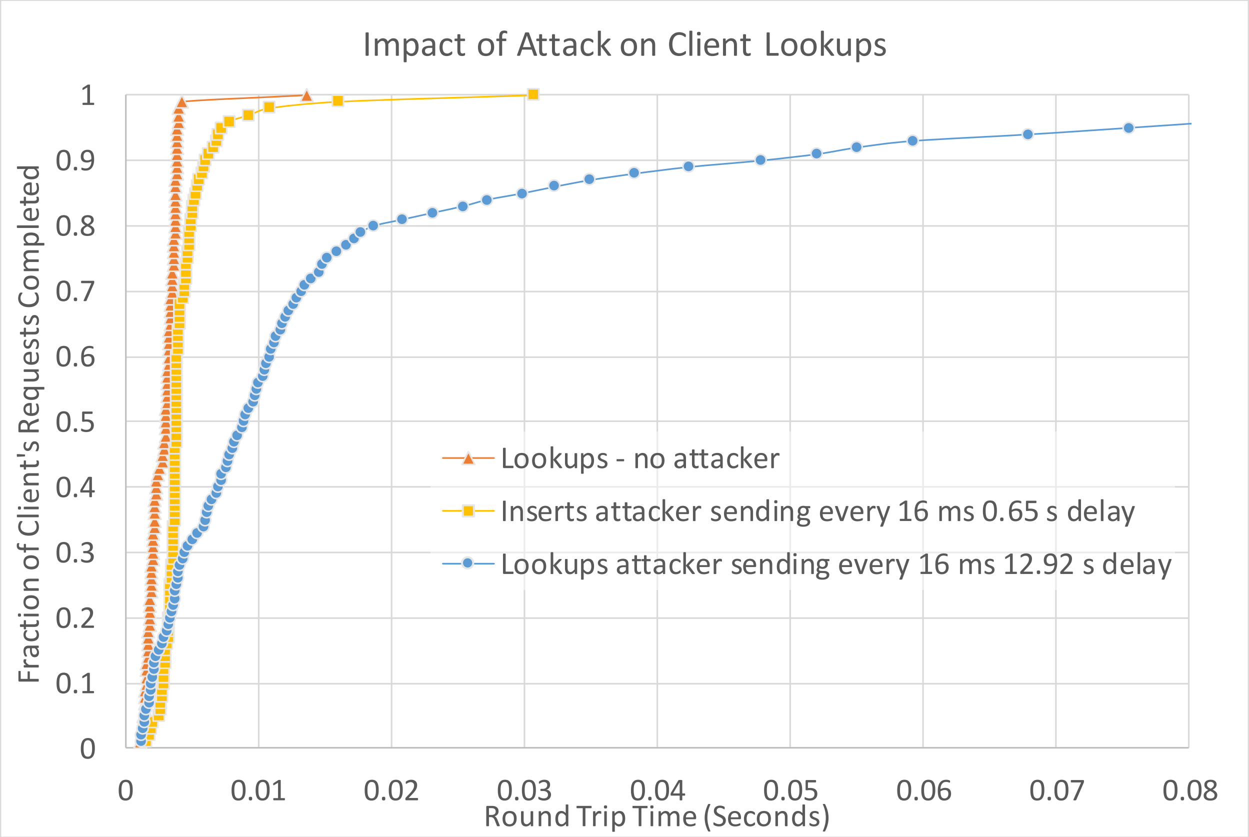

Figure 15. Cumulative Distribution Function of Client Round Trip Times with Lookup Requests. This graph shows one trial where all GNRS servers were behaving normally and the client looked up each of the 1,000 GUIDs it inserted. Then, inserts and lookups were done in a second trial with the attack server sending an insert every 16 ms.

5.4 Tradeoffs

Computer system design involves tradeoffs. Sometimes, by fixing one problem, a software engineer introduces more. In this section, we discuss four possible approaches to preventing the attack demonstrated in 5.3 above, starting with approaches that would decrease the system's mobility functionality. These options, while feasible, would restrict DMap's ability to meet the mobility design goals of MobilityFirst, a tradeoff we consider to not be worthwhile. Since the attack involves a malicious DMap server that could ignore GUID authentication, we do not consider that as a defense for this attack.

A key part of this attack is that the requests coming from the malicious server are recursive, as overviewed in Section 5.1. One way this type of attack could be prevented is for all DMap servers to check that every received recursive request comes from devices in their AS. Then, when the attacking server sends the recursive requests to the other DMap servers, they would note that it is a recursive request with an external source and disregard it. However, if DMap servers were to ignore all "external" recursive inserts, the DMap servers would have to know who all of the first level clients are, and not allow new ones. This would restrict devices moving between ASs, and not meet MobilityFirst's mobility goal. One could look into requiring a lower level handshake first to establish presence in an AS before being inserted into the GNRS.

Another option would be to only allow a fixed number of recursive inserts from each Network Address in a given time period. Unfortunately, this would limit how often devices could move. Additionally, it would require additional per-Network Address state on each server to count the number of recent inserts. This could be ineffective against the attack, since the attacker sent inserts at rates three orders of magnitude slower than the client. If it was effective, the attacker could bypass this measure by spoofing its Network Address or rotating through which other DMap servers it sends all of the requests to, instead of just sending each insert request to all other DMap servers.

DMap might combat this attack by allowing more than one “local server” to serve each client. An attacker could still flood with insert requests, but would have to either find a GUID that would be inserted into all local servers serving the target client, or alternate between the GUIDs it is inserting. If the “backup” local GNRS server is well known, an attacker can find such GUID(s) easily. Since keeping secrets as a means of security is well-understood to be a weak approach, doing so is not a viable alternative to the backup server being well known.

As discussed in the previous section, client Insert Requests are not the only type of communication getting dropped from the target server’s queue. When successful insert responses from remote servers dropped by queue, the initial server send them again, and eventually waits for a timeout before declaring the Insert (partially) successful. One option would be to use a priority queue, giving insert response packets a higher priority than new insert packets. If work has already been done to insert a GUID, that should be recorded; otherwise, the work will have to be done again. This does not solve the problem of insert requests getting dropped, it merely increases the likelihood that Insert Requests that have begun are successfully completed so that they are not resent by the client.

5.5 Summary

In this chapter, we demonstrated a denial of service flooding attack on the DMap Global Name Resolution Service system. The viable approaches to prevent this attack, filtering and rate-limiting Insert Requests, both involve a tradeoff with the support for mobility in the Global Name Resolution service.

6. Effects of Transport Layer Protocol on GNRS Design

A Global Name Resolution System would lie in the application layer of the Internet Protocol Stack. Below the application layer is the Transport Layer. Applications need to interface with one or more protocols in the transport layer in order to successfully move information around the Internet. Two widely used Transport Layer protocols are User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). UDP is a very lightweight protocol for stateless best-effort communication. TCP is a reliable, ordered transport layer protocol. When using TCP, a connection is established between each client-server pair via a three-way handshake, shown in Figure 16. First, a synchronization packet is sent from the client to the server. That transmission is acknowledged by the server to the client and sent with a synchronization number from the server. Finally, the client acknowledges the server's SYN and sends data.

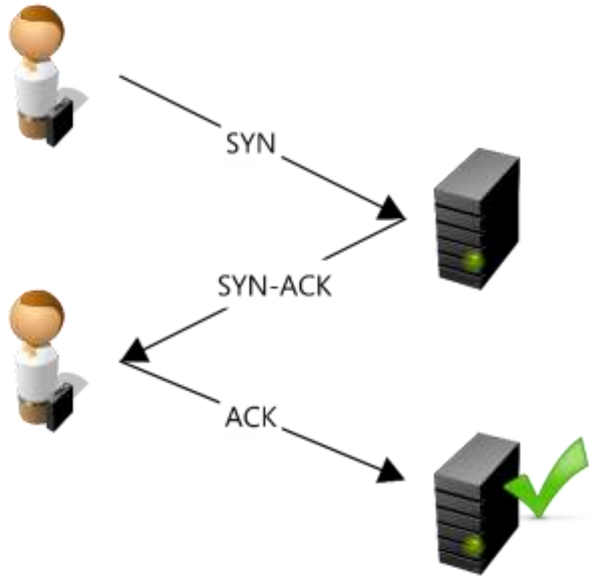

Figure 16. ***TCP Three-Way Handshake.*** *[21]*

UDP Insert Message	TCP Insert Message
70 bytes of data	70 bytes of data
14 bytes Ethernet frame	14 bytes Ethernet frame
20 bytes IPv4 header	20 bytes IPv4 header
8 bytes UDP header	20 bytes TCP header
112 bytes on wire	124 bytes on wire

Table 4. Comparision of Minimum UDP and TCP Header Size

When using UDP, unless a response is sent at the application layer, the client does not know if the server received its request. The request could have been dropped, or may just be taking a long time to reach the relevant remote servers.

As discussed in Section 4.2, Auspice's replica-controllers assume little contact compared to active replicas, and would be susceptible to a voluminous attack. It would be easier for them to throttle TCP flows than UDP packets. Replica-controllers would have a session open with each TCP client and can measure its rate of requests, and automatically ignore clients who request new information at unreasonable, attack level rates.

There are two opportunities for DMap to verify that an insert request actually came from the GUID requesting insertion: the local DMap server that initially receives the request, and the (likely remote) DMap servers responsible for hosting the mapping. Many denial of service attacks rely on the ability to spoof a target's IP address to a local DMap server. Regardless of GNRS design, it is easier to pretend to be at a different address than when you are using UDP because a connection is never established, the network just routes packets to the address on the header. Spoofing an address is significantly more difficult when using TCP as an underlying protocol, because the client needs to receive the SYN-ACK at the IP address they sent the server. This could be circumvented by using a man-in-the-middle attack, during which the attacker sits on the path between the client and the server and impersonates each to the other. Since DMap uses UDP, no man-in-the-middle attack is necessary to spoof targeted GUIDs as in the attack demonstrated in Chapter 5.

If DMap was using TCP, it would be less important to verify insertion request data upon insertion, because the data is likely to have been verified by the initial DMap server receiving the request. However, even though DMap runs on UDP, when a DMap server responsible for hosting GUID G_X's mapping receives a request to insert GUID G_X at some network address(es), the server does not check to see if GUID G_X is reachable at the bindings it claims it is located at. Although this means that information in DMap may not be accurate, this is a justified tradeoff. If the servers did check for connectivity at the given bindings, a reflection attack would be possible. In a reflection attack, a malicious device sends an insert request to the server, claiming to be at the target's address. Then, since the server would check that the aforementioned address is valid, the target receives unwanted traffic. This wastes the target's bandwidth, which may be limited, especially in cases of rapid mobility that MobilityFirst was designed to support.

7. Future Work

Testing under a realistic topologies and mobility model(s) remain a consideration for future work. The attack mentioned in Section 4.3, where migration messages are forged, could also be tested. It was not tested at this time because Migration Messages are not currently implemented. Additionally, instead of a constant request rate for sequential GUIDs, a Traffic Model using a Zipf distribution to model request rates for GUIDs would be more realistic and representative of actual Internet traffic.

7.1 Topology

The GNRS systems could be evaluated under a more geographically realistic network topology, using GeoTopo [22] to model the network topology. Additionally, one could compare how DMap and Auspice function across multiple topologies, instead of imposing a specific topology on them. Network architecture and network topology can influence each other's evolution. Similarly, traffic loads and patterns may be different (in the immediate sense and as the Future Internet Architecture changes over time) based on the underlying network architecture.

In the grid on ORBIT, the nodes are in a small room and all within radio range of each other, creating a mesh network, where each node is directly connected to each other node. It is possible to create a multi-hop topology in such an environment by injecting noise as in [23], but this is not functioning on ORBIT at the time of this publication. This noise injection system should be employed to prevent testing on a trivial "mesh" where each device is exactly one hop away from each other device.

7.2 Mobility Model

Supporting mobility is at the core of MobilityFirst, and incorporating an appropriate mobility model is the next logical step for this study. The mobility model need only take into account mobility that would change a GUID's NA. One advantage of name records mapping a GUID to an AS number is that when a GUID moves within an AS, its mapping (G_X, AS 1) does not change. This means the system resources are not strained, or even used, when devices move within a network. As Yang et al. observed in [24], most device movement is within one AS. Unfortunately, we were unable to observe this in action as the DMap code does not have client-side updates implemented at this time.

Since a NA is comparable to an AS in today's Internet, we found a model derived from observing when users connected and disconnected to different ASs. Specifically, a Hidden Markov model derived in "Measurement and Modeling Study of User Transitioning Among Networks," gives a transition probability matrix of a user's next state based on their current state [24, 25]. They include six states: (1) connected to zero networks (2) connected to zero new networks and one total network, (3) connected to one new network, (4) connected to multiple networks, none of which are new, (5) connected to multiple networks, one of which is new, and (6) connected to multiple new networks. Their model discretizes mobility among networks in discrete-time chunks of 15 minutes. In order to prevent "batch" movement in experiments, whenever the model dictates that a user changes state during a 15-minute period, the exact time during the 15-minute period could be randomly chosen.

Probability of Transitioning to Each State Given Current State

Current State	0 new, 0 total	0 new, 1 total	1 new, 1 total	0 new, >1 total	1 new, >1 total	>1 new, >1 total
0 new, 0 total	0.88577709	9.78E-06	0.11012325	0	0	0.00408987
0 new, 1 total	0.20181485	0.7266925	0.02349103	0	0.04486134	0.00314029
1 new, 1 total	0.6474509	0.2834485	0.04665651	0	0.01936993	0.00307416
0 new, >1 total	0.0749543	0.43327239	0.01005484	0.42413163	0.05393053	0.00365631
1 new, >1 total	0.17307092	0.59418932	0.01968135	0.15901281	0.052796	0.00124961
>1 new, >1 total	0.4380704	0.40026076	0.03976532	0.07887875	0.03976532	0.00325945

Table 5 Transition Probability Matrix [24, 25]

Additionally, one may refer to [26], which quantitatively compares different network architectures that support mobility and discusses mobility of devices across ASs.

8. Conclusion and Comparison to Today's Internet

8.1 Suggestions

In this section, we summarize suggestions to make the GNRS designs more secure.

8.1.1 Auspice Suggestions

Instead of automatically using Paxos for all GUIDs, Auspice could only use it for GUIDs that recursively represent groups of terminal GUIDs, which are more likely to have operations by many actors that need to be performed in the same order. Allowing variety among mappings for non-recursive GUIDs would allow Auspice, or the GUIDs themselves, to customize mappings based on where they are stored. As discussed in Section 3.4, this feature could increase effective communication with multi-homed devices.

Another suggestion or design discussion regarding Auspice is the assumption that replica controllers will be contacted on a limited basis. Assuming this assumption is valid, it should be enforced programmatically.

Finally, as discussed in Section 4.3.1, an attacker could just try to expand name records to push past the storage capabilities of Auspice servers' databases by setting arbitrary (key, value) pairs.

Vu et. al. address similar concerns in DMap by defining name records as simple GUID-NA mappings and claiming that each GUID can be multi-homed at up to 5 NAs [9].

8.1.2 DMap Suggestions

DMap stores much less per-GUID data than Auspice, which prevents the kind of attacks discussed in the previous section. However, DMap servers would benefit from keeping an additional bit of data per GUID: a Boolean value indicating whether the server is acting as a Deputy Autonomous System for that GUID's mapping. This is one example where the tradeoff between storage and a computation-intensive operation is clear. Keeping an additional true/false value would require minimal additional storage as it is literally a bit of information. The alternative to looking up this bit is the server that received a migration message for GUID G_X computing whether it is a deputy for G_X. This requires hashing the GUID at least *M* times, per the IP hole protocol[5]. Since the server would have to look up the name record for G_X to see if the server even has a name record for G_X to drop, looking up the deputy true/false value adds no additional computation time.

As discussed in Section 3.1.1, DMap's local replication specifies the use of an additional hash function within each AS to determine a local server on which to store mappings of GUIDs within the AS. During lookups, this prevents AS 1 from having to contact a remote AS to learn that GUID G_X is located within AS 1. In case G_X is not in AS 1, AS 1 simultaneously sends requests to the local server and to a "global" copy, using one of the original *K* hash functions.

Since intra-AS routing is not stipulated by GNRS designs, intra-AS replication should not be either. For G_Z located in AS 1, DMap leaves intra-AS routing – routing from AS 1's gateway router to G_Z's device - up to AS 1. AS 1 will already have to have a routing table to determine where each device is located within the AS. Since this is the case, DMap should leave local replication up to the ASs.

As discussed in Section 4.1.1, DMap may fail to meet an availability goal for certain GUIDs. For some GUIDs, the independent hash functions output IPv4 addresses hosted by the same AS,

[5] It may take more than *M* hashes if the server is not sure which of the *K* hash functions lead to the IP hole. Assuming at most one hash function leads to an IP hole it would take *K* + *M* hashes in the worst case.

resulting in one fewer placement of the GUID's name record. There are at least two simple solutions to this issue. First, when a mapping overlaps, DMap servers could take the output of one of the hash functions and follow the IP hole protocol. Alternatively, the number of hash functions used to determine ASs could be increased, either by designating an extra "backup" hash function, or increasing the system parameter *K*. GMap reduces the likelihood of such AS collisions in yet another way. GMap explicitly checks that the hosts picked within each level (global, local and regional) do not collide with hosts chosen in another level. The only possible server-collisions are among the replicas at each geographic level.

8.1.1 GMap Suggestion

While GMap's insert algorithm offers a benefit over DMap, its lookup policy of first searching for GUID name records near the requester is futile when the GUID is actually located far away. This process would be more successful if regional and local caches were designated where the name records are expected. As demonstrated in Figure 6, when GUID Y wants to look up GUID X, GMap first checks the servers local to Y that X would be stored on *if X were in the same geographic location as Y*. Since these same servers are checked for any GUID in Y's city or region that is looking up GUID X, it makes sense to have a copy of G_X's name record at these servers. If G_X's name record were cached on these servers, when X could be quickly resolved in Y's region. However, under the current system design, these servers would only have X's name record in their cache by the happy accident of being on the path traversed by a successful lookup request for G_X.

As mentioned above, our goal was to discern the features of MobilityFirst's architecture that allow continued communication during node mobility through examining denial of service attacks in proposed global name resolution services.

8.2 Comparison to Current Internet

Routing on the Internet is based on Internet Protocol (IP) addresses. IP addresses were designed when devices were connected to the Internet via a wired connection, such as Ethernet. The IP address uniquely identified the device connected via the wire to the network, and also act as a routable address used by routers to determine the next hop on a packet's way to its destination IP address. Today, devices may be connected to multiple IP addresses at once, or wirelessly switch between IP addresses while communicating with other network devices. In the current Internet, devices that are more mobile, such as personal computers and laptops, tend to have less traffic than static devices. Total mobile data traffic (including smartphones, tablets and mobile PCs) is 5.3 EB per month, while total fixed data traffic is 60 EB per month [27].

However, there are certain instances where an extremely popular, high traffic source is moving. For example, consider the Super Bowl, where many people are tuned in to a streamed source moving between many different cameras. Here, there could be a GUID representing the main Super Bowl broadcast that would relocate rapidly.

IP forms a flexible base for today's Internet, with more features and constraints built on top of it. It is a "best effort" system that is designed to be simple and approximate. As complexity rises, so do opportunities for security breaches and DoS attacks. With a best effort system, one can assume something(s) go wrong, and that's tolerable in the system. Instead, if one is trying to keep a system perfectly secure and work in every eventuality and edge case, it becomes too cumbersome to be effective. Security-wise, it can turn into an attack/patch war. Similarly, as a basic service of Mobility First, the GNRS should have as few constraints as possible. Key principles of computer system design are simplicity and modularity. Adding additional protocols on top of the GNRS would allow for more advanced features and could incorporate more constraints on how they are used, while leaving the GNRS as a simple primitive.

Auspice struggles more in this respect, trying to keep per-GUID statistics and do per-GUID load balancing, which can be computationally expensive and use a lot of storage. This would likely require a lot of fine-tuning on the length of epochs. Also, Paxos may be too close to trying to be "perfect" with an all-or-nothing update approach.

A major difference between today's Internet and MobilityFirst's network is the target endpoint, an IP address and a Network Address, respectively. "Bringing down" (ie. clogging) a Network Address affects more devices than an IP address, but is harder to do because the link capacity would be greater. Handley and Rescorla discuss denial of service attacks in the current Internet [28]. Like many denial of service attacks in the current Internet, and setting aside the basic advice of securing system-critical servers, we leave the attack demonstrated in Chapter 5 unsolved. In order to meaningfully address the malicious server attack on a target server in the DMap implementation of a Global Name Resolution Service, the main design goal of supporting rapid mobility would be compromised as it would restrict either which GUIDs could register with a DMap servers or how frequently they could change their name record. The Global Name Resolution Service has some fault-tolerance build in, having *K* replicas of each GUID precisely for robustness to failures, including denial of service attacks. With enough replication, the GNRS system designs proposed are, like the current Internet, likely to function “well enough”.

9. References

[1] K. R. Sollins, D. D. Clark, W. Lehr, J. Kurose, A. Venkataramani and S. Bauer, "NeTS: Large: Collaborative Research: Location-independent Networks: Evaluation Strategies and Studies," Submission to the National Science Foundation, Solicitation 13-581, November 19, 2013.

[2] P. Gasti, G. Tsudik, E. Uzun and L. Zhang, "DoS and DDoS in named data networking," *International Conference on Computer Communications and Networks,* pp. 1-7, 2013.

[3] A. Compagno, M. Conti, P. Gasti and G. Tsudik, "Poseidon: Mitigating Interest Flooding DDoS Attacks in Named Data Networking," in *Annual IEEE Conference on Local Computer Networks*, Sydney, Australia, 2013.

[4] K. Wang, J. Chen, H. Zhou, Y. Quin and H. Zhang, "Modeling denial-of-service against pending interest table in named data networking," *National Engineering Laboratory for Next Generation Internet Interconnection Devices,* July 2013.

[5] A. Venkataramani, J. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao and S. Banerjee, "MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture," *ACM SIGCOMM Computer Communication Review,* 2014.

[6] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang and B. Zhang, "Named Data Networking," *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 3, pp. 66-73, July 2014.

[7] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," *SIGCOMM Computer Communications Review,* vol. 34, no. 2, pp. 39-53, 2004.

[8] T. Peng, C. Leckie and K. Ramamohanarao, "Survey of Network-Based Defense Mechanisms Countering the Dos and DDoS Problems," *ACM Computing Surveys,* vol. 39, April 2007.

[9] T. Vu, A. Baid, Y. Zhang, T. D. Nguyen, J. Fukuyama, R. P. Martin and D. Raychaudhuri, "DMap: A Shared Hosting Scheme for Dynamic Identifier to Locator Mappings in the Global Internet," in *IEEE 32nd International Conference on Distributed Computing Systems*, Washington, D.C., 2012.

[10] I. Seskar, "MobilityFirst Wiki," GNRS Installation, 7 July 2015. [Online]. Available: http://mobilityfirst.orbit-lab.org/wiki/Proto/cModules/b0GNRS/c0Installation#no1. [Accessed 24 May 2016].

[11] Sharma, A, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook and A. Yadav, "A Global Name Service for a Highly Mobile Internet," UMASS Computer Science Technical Report: UM-CS-2013-023, 2013.

[12] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook and A. Yadav, "A Global Name Service for a Highly Mobile Internetwork," *ACM SIGCOMM,* 2014.

[13] A. Venkataramani, "MobilityFirst GNS on Github: Getting Started," 5 July 2016. [Online]. Available: https://github.com/MobilityFirst/GNS/wiki/Getting-Started. [Accessed 8 July 2016].

[14] A. Venkataramani and D. Westbrook, "MobilityFirst/gigapaxos: GigaPaxos is a group-scalable reconfigurable consensus system that can be used to efficiently and easily manage a very large number of independent lightweight replicated state machines.," GitHub, Inc., 14 July 2016. [Online]. Available: https://github.com/MobilityFirst/gigapaxos. [Accessed 14 July 2016].

[15] Y. Hu, R. D. Yates and D. Raychaudhuri, "A Hierarchically Aggregated In-Network Global Name Resolution Service for the Mobile Internet Technical Report," WINLAB-TR-442, North Brunswick, NJ, March 2015.

[16] J. P. Fernández, "Two-in-one DNS server with BIND9," 20 February 2006. [Online]. Available: https://pupeno.com/2006/02/20/two-in-one-dns-server-with-bind9/. [Accessed 7 August 2016].

[17] C. T. Rock, "crockct/GNRS-DOS on GitHub," 8 August 2016. [Online]. Available: https://github.com/crockct/GNRS-DOS. [Accessed 3 September 2016].

[18] "Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT)," [Online]. Available: http://www.orbit-lab.org/. [Accessed 16 December 2015].

[19] T. Rakotoarivelo, M. Ott, G. Jourjon and I. Seskar, "OMF: a control and management framework for networking testbeds," *ACM SIGOPS Operating Systems Review,* vol. 43, no. 4, pp. 54-59, Jan. 2010.

[20] S. Goldberg, "Why Is It Taking So Long to Secure Internet Routing?," *ACMqueue,* vol. 12, no. 8, September, 2014.

[21] "Wikimedia Commons," 10 March 2009. [Online]. Available: https://commons.wikimedia.org/wiki/File:Tcp_normal_2.png. [Accessed 21 August 2016].

[22] Y. Hu, F. Zhang, K. K. Ramakrishnan and D. Raychaudhuri, "GeoTopo: A PoP-level Topology Generator for Evaluation of Future Internet Architectures," in *Proceedings of the 23rd IEEE International Conference on Network Protocol (ICNP)*, 2015.

[23] S. K. Kaul, M. Gruteser and I. Seskar, "Creating Wireless Multi-hop Topologies on Space-Constrained Indoor Testbeds Through Noise Injection," in *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, Barcelona, 2006.

[24] S. Yang, J. Kurose, S. Heimlicher and A. Venkataramani, "Measurement and Modeling Study of User Transitioning Among Networks," in *IEEE INFOCOM 2015 conference*, 2015.

[25] S. Yang, Interviewee, *Personal Correspondance.* [Interview]. 20 May 2016.

[26] Z. Gao, A. Venkataramani, J. Kurose and S. Heimlicher, "Towards a Quantitative Comparison of the Cost-Benefit Trade-offs of Location-Independent Network Architectures," in *SIGCOMM*, Chicago, IL, 2014.

[27] "Ericsson Mobility Repot: On the Pulse of the Networked Society," Ericsson, Stockholm, Sweden, November 2015.

[28] M. Handley and E. Rescorla, Eds., Internet Denial-of-Service Considerations, IETF, November 2006.

[29] K. R. Sollins, "TWC: Small: The Interaction Between Mobility and Denial-of-Service Attacks," Solicitation: National Science Foundation, NSF 15-575, SaTC Small, November 18, 2015.

[30] I. Stoica, D. Karger, M. F. Kaashoek and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol," in *SIGCOMM*, New York, NY, 2001.

[31] A. Venkataramani and D. Westbrook, "MobilityFirst/GNS: A Global Name Service for a Highly Mobile and Secure Internetwork," GitHub, Inc., [Online]. Available: https://gns.name/wiki/index.php?title=MobilityFirstGit.

[32] "Rainbow table," Wikipedia, 25 July 2016. [Online]. Available: https://en.wikipedia.org/wiki/Rainbow_table. [Accessed 3 August 2016].

[33] I. Seskar, "ORBIT: GNRS Management," June 2015. [Online]. Available: http://www.orbit-lab.org/wiki/Other/Summer/2015/cMF5. [Accessed 28 November 2015].