Experiences Building Security Applications on DHTs

Roxana Geambasu* Jarret Falkner* Paul Gardner† Tadayoshi Kohno*
Arvind Krishnamurthy* Henry M. Levy*

*University of Washington and †Vuze, Inc.

Abstract

In the recent past we introduced two new security applications built on peer-to-peer systems and distributed hashtables (DHTs). First, we designed Adeona [18], which leverages DHTs to provide a privacy-preserving laptop tracking solution. Second, we designed the Vanish [10] self-destructing data system, which uses DHTs to protect against retroactive attacks on archived data in the cloud. Both systems exploit intuitive properties of DHTs that differentiate them from centralized solutions: *e.g.*, complete or partial decentralization, giant scale, and geographic distribution. We implemented and made publicly available research prototypes of both Adeona and Vanish; the Adeona prototype uses OpenDHT as its underlying DHT and the Vanish prototype uses the Vuze DHT.

While the properties of DHTs make them a tempting environment for new security-based systems, existing DHTs were never designed to support security or privacy applications, and such applications therefore stress DHTs in new ways. This paper provides a retrospective from our collective experience both designing and prototyping the two DHT-based security/privacy applications, and operating and designing deployed DHTs (OpenDHT and Vuze). We discuss limitations and vulnerabilities of modern DHTs for security applications and propose very simple defenses that — perhaps surprisingly — greatly raise the bar for existing DHTs against certain privacy attacks. We also advocate for a hybrid approach that combines the best of both decentralized DHTs and centralized services for new security applications. Our goal is to inform the design of future DHTs and to strengthen the applicability of existing DHTs for supporting applications such as Adeona and Vanish.

1 Introduction

Over the last decade, Distributed Hash Tables (DHTs) have been extensively explored in the research community and deployed for numerous applications on the Internet. At its core, a DHT simply implements the services of a remote hash table through internal coordination among nodes in a global-scale peer-to-peer (P2P) network. DHTs have many useful properties: they're highly distributed and scalable, they're decentralized and self managing (*i.e.*, there is no single trusted entity in charge), and they are simple to use. Moreover, DHTs are a natural structure for information sharing among independent parties in a large distributed environment.

Recently, several projects have used DHTs as the basis for security- and privacy-based applications. We have developed two projects in this space: Adeona, a privacy-preserving method for tracking lost or stolen mobile devices [18], and Vanish, a method for creating self-destructing digital data [10]. To explore our research concepts, we designed, prototyped, and released both systems to encourage and facilitate the continued analysis, experimentation, and improvement of these systems by ourselves and others.

Adeona and Vanish are interesting because their security and privacy requirements stress P2P systems and DHTs in new ways. Existing wide-scale DHTs are used mostly for file sharing ap-

plications, where the definition of security for the DHT is different than for security and privacy applications. While there was been significant research in DHT security in the past, most efforts have focused on byzantine attacks (*e.g.*, disrupting the DHT and routing attacks) [23] and node-crawling attacks (*e.g.*, to identify membership) [26]. In contrast, security applications are more concerned with *information harvesting* attacks. An information harvesting attack might reveal to an attacker the contents of private data stored in the DHT. But an information harvesting attacker might seek other information as well, such as the IP addresses of the clients that originally inserted the data into the DHT or that later retrieved that data. Such attacks are fundamentally different than traditional attacks and we believe require a new perspective in DHT design.

In designing prototypes for Adeona and Vanish we chose to use existing, currently-deployed DHTs. The Adeona prototype builds upon OpenDHT [17] — a closed-access research system running on several hundred PlanetLab nodes around the world. In contrast, the Vanish prototype builds upon the Vuze DHT [2] — an open-access, commercially-supported production system that runs on over one million privately-owned nodes around the world [8]. At the time we implemented each system we believed that the corresponding DHT (OpenDHT for Adeona and Vuze for Vanish) best met the needs of each system.

In this paper we seek to reflect upon our experience of designing and deploying these two DHT-based security applications. Our goals for this paper are threefold:

- First, we provide perspective on the design of Adeona and Vanish. In particular, we discuss the security needs for each system and why we chose DHTs to support them. We then re-evaluate OpenDHT and Vuze from a security viewpoint and identify those design and implementation properties that, in retrospect, either enhanced or inhibited our applications.
- Second, we define a set of crucial resistance properties for future DHTs that can strengthen them against information privacy attacks. As well, we present and evaluate a number of simple enhancements that can strengthen existing DHTs such as Vuze against these attacks.
- Third, we discuss techniques for modifying DHT-based security applications themselves to enhance application-level security.

In addition to our involvement with Adeona and Vanish, we build on our experience with DHTs as well. The co-authors of this paper include two of the designers of Adeona (A.K. and T.K.), three of the designers of Vanish (R.G., T.K. and H.L.), the current maintainer of OpenDHT (A.K.), and the designer of the Vuze DHT (P.G.).

Overall, we hope that our lessons and experience will inform the design of future DHTs and DHT-based security applications. Our evaluation shows that while existing DHTs provide attractive properties for applications like Adeona and Vanish, they also create serious security challenges in their current state. Given that the needs of security applications are different than those of standard DHT applications, it's not surprising that DHTs have security weaknesses. But, perhaps somewhat surprisingly, we find that simple changes to existing DHTs like Vuze can significantly raise the bar against attackers. Finally, we argue that basing the security of an application on a single DHT alone may be too risky given the current state of the art in deployed DHTs. We therefore explore an architectural strategy in which a security application can benefit from the properties of DHTs (global scale, decentralization, etc.) while also minimizing the risks associated with relying upon a single DHT.

The remainder of our paper is organized as follows. The next section provides background on Adeona, Vanish, OpenDHT, and Vuze. Section 3 reflects on the design of Adeona and Vanish, and in particular, on the properties of the DHTs used for their prototypes, what worked, and what did not. In Section 4 we describe requirements for DHTs supporting security applications and present a number of simple strategies for meeting those requirements in the context of existing DHTs.

Section 5 describes application-level strategies for tolerating DHT weaknesses. We discuss related work in Section 6 and Section 7 summarizes our results.

2 Two DHT-Based Security Applications

Adeona and Vanish are new security applications that strive to achieve privacy properties not met by previous systems. Our quest to achieve these challenging properties motivated us to build on top of DHTs. This section describes these two new applications as well as their underlying DHTs, OpenDHT and Vuze.

2.1 Adeona

In recent years, theft of mobile devices has become a serious problem, with about one in ten laptops stolen within the first 12 months of purchase and 97% of them never recovered. While there exists device tracking software to combat this growing problem [13], most solutions fail to satisfy the privacy goals of honest users. For instance, clients typically send their identity and current location information (such as IP address, GPS location, and so on) to a storage server run by the tracking service to aid in recovery when the device is lost or stolen. The tracking service can then monitor the location of the device even while it is with the owner, thereby compromising the owner's location privacy.

To address such privacy concerns, we designed Adeona, a *privacy-preserving device-tracking system* that provides strong guarantees of location privacy for the device owner's legitimately visited locations, without compromising the ability to track a stolen [18]. Adeona was designed after a careful examination of an intricate threat model that yielded the following privacy goals.

- Updates sent by the client must be *anonymous* and *unlinkable*. This means that no adversary — including the remote storage provider — should be able to either associate an update with a particular device, or even associate two updates as being from the same device.
- State maintained on the client must exhibit *forward-privacy*, meaning a thief, even after seeing all of the internal state of the client, cannot learn past locations of the device.
- Searches performed on the remote storage must also exhibit *forward-privacy*; the owner should be able to efficiently search the remote storage without revealing to the storage provider past locations of the device.

Adeona achieves these privacy goals by a combination of cryptographic mechanisms and its use of a DHT for remote storage. The client application utilizes a forward-secure pseudorandom generator [3] (FSPRG) to efficiently and deterministically generate the cryptographic keys used for the updates, rendering the updates anonymous and unlinkable. FSPRG also ensures forward-privacy when the device is stolen. A thief, after determining all of the internal state of the client — such as the current FSPRG values — cannot use Adeona to reveal past locations of the device. The owner can efficiently search the remote storage for the updates starting with the most recent ones, without revealing the keys associated with the earlier updates. Further, the updates are scheduled at pseudorandomly determined times, which minimizes the remote storage service's ability to link updates based on timing information. Finally, Adeona uses a DHT as the back-end storage provider. Using a DHT for storage means that data storage requests do not pass through any single trusted component and eliminates privacy and security concerns that arise out of having a single entity observing and storing all updates in the system. If all client updates passed through a central service instead of a distributed system like a DHT, then that central service would still be able to profile users. For example, by recording when a residential IP address sends updates, the central

service could still learn partial information about whether a user was home at a given period of time or not.

We implemented a proof-of-concept Adeona prototype on top of OpenDHT. We chose OpenDHT because it provides a distributed storage service over hundreds of PlanetLab machines around the world, because it offers data persistence for up to one week, and because of the active research community based around OpenDHT.

2.2 Vanish

Vanish addresses a significant challenge to personal data privacy raised by the move to Web services and the cloud. This transition is causing huge volumes of personal data to be stored, cached, and archived by third-party services, sometimes without the user's knowledge. The result is that users have no control over the lifetimes of their data; i.e., a user cannot be sure that her data is destroyed, even if she explicitly deletes it, and in fact, even if she terminates her account with a Web service provider. Therefore, past archived data may endure potentially forever and become a risk for exposure through accidental means, mismanagement, data theft, legal proceedings, and so on.

To address this data privacy concern, we designed Vanish as a *self-destructing data system* in which *all* copies of data would become permanently unreadable at a user-specified time [10]. Our goals for self-destructing data were quite strong, e.g., we want data to self-destruct: (1) even if an attacker retroactively obtains a copy of the data and any relevant cryptographic keys from before the timeout; (2) without the use of any explicit delete action by any parties involved; (3) without needing to modify any of the copies of the data; (4) without the use of secure hardware; and (5) without relying on new and trusted external services.

Our prototype attempts to meet these goals through the use of cryptographic techniques plus a decentralized, global-scale DHT. At its core, Vanish encrypts a user's data with a random encryption key that is never revealed to the user; it then breaks the key into pieces and sprinkles those pieces across random nodes in the DHT, using Shamir secret sharing [20]. For some period of time, given the encrypted object and the indices of the random nodes, it's possible to retrieve enough keys to decrypt the data. However, after some time, the keys disappear due to churn in the DHT or purging of local stores by DHT nodes. The decentralized nature of the DHT (e.g., nodes might be distributed over international boundaries), plus the nature of DHT addressing (nodes may leave and rejoin at a different address), would make it very difficult for an attacker months or years in the future to retrieve the keys from the past.

We implemented a proof-of-concept Vanish prototype on top of the Vuze DHT. We chose Vuze mainly because of its size, its global distribution, and the lack of central control over individual DHT nodes. As in the case of Adeona, using a DHT means that there is no single trusted third party responsible for the privacy of the user's data or its deletion.

2.3 Two Leading DHTs

We now provide some additional background on both OpenDHT and Vuze.

OpenDHT Background. The goal of OpenDHT (originally named OpenHash) was to provide a free, public DHT service that runs on PlanetLab [12, 17]. This motivational quote from the original OpenHash paper is illuminating (use of italics has not been changed) [12]:

> [W]e issue a call-to-arms to deploy an *open, publicly accessible* DHT service that would allow new developers to experiment with DHT-based applications *without* the burden of deploying and maintaining a DHT. We believe that there are many simple applications

that, individually, might not warrant the effort required to deploy a DHT but that would be trivial to build over a DHT service *were one available*.

OpenDHT met this goal from the time of its creation through to mid-2009, and Adeona was one of the applications that benefited from OpenDHT. Internally, OpenDHT consists of hundreds of P2P nodes running on PlanetLab. Externally, each of these nodes expose a public-facing RPC interface. Using this public RPC interface, a client can ask an OpenDHT node to store a chunk of information (up to 1KB in size) keyed off an arbitrary 160-bit index of the client's choosing. A client can also query a (possibly different) OpenDHT node for the value stored at a given 160-bit index. OpenDHT uses a number of techniques to internally distribute and make data available within the DHT.

The separation between OpenDHT nodes and client nodes is important for OpenDHT. There are no restrictions on who can be a client node. This is necessary to support OpenDHT's goal of being a public service DHT. On the other hand, there are strong restrictions on who can actually participate as an OpenDHT node. A node can only join the DHT itself if it knows a necessary secret key. In practice, this means that all nodes in OpenDHT are under the control of the OpenDHT maintainer.

In mid-2009 the original maintainer of OpenDHT announced that he was taking down OpenDHT. Because Adeona depended on OpenDHT, one of us (A.K.) has now taken on the maintenance role of OpenDHT.

Vuze Background. Vuze (*a.k.a.* Azureus) is a popular BitTorrent P2P client with around 1.5 million users active at any one time and a user base of over 4 million users. One of us (P.G.) originally developed the Vuze DHT as a simple backup solution when a BitTorrent tracker was unavailable. A client unable to contact a tracker could advertise details in the DHT and other clients could look them up. The Vuze DHT gradually evolved to support many more features, including fully trackerless mode for torrents, ratings and popularity calculation for torrents, and even an instant messaging system. Although security has always been a concern for the Vuze design, its focus has largely been on attacks aimed to disrupt the network or hurt data availability or persistence, data pollution and modification, and overloading of popular nodes. However, protection against data-gathering attacks was never a goal, because for the main Vuze application (tracking), revealing tracking associations was not considered a privacy risk. Some simple protection against DoS attacks was inserted, namely the possibility of limiting the number of ports behind an IP address, but this was never enabled because such attacks never materialized.

The Vuze DHT is based on the Kademlia [15] protocol. Each Vuze DHT node is assigned a "random" 160-bit ID based on its IP and port, which determines the index ranges that it will store. To store an (index, value) pair in the DHT, a client looks up 20 nodes with IDs closest to the specified index and then sends `store` messages to them. Vuze nodes republish the entries in their cache database every 30 minutes to the other 19 nodes closest to the value's index in order to combat churn in the DHT, assuming that they have not themselves recently received a push of those values. Nodes also receive replicated values for their index space with they join the DHT. Nodes further *remove* from their caches all values whose `store` timestamp is more than 8 hours old; this timeout has been extended to up to 3 days to support Vanish clients.

3 Examination of Storage Solutions for Security Applications

We now examine our experiences building security applications on top of DHTs, both from our perspectives as the designers of Adeona and Vanish and from our perspectives as the maintainers of their respective DHTs.

ADEONA Primary goal: location privacy						
Storage System Requirements	Storage solution					
	Centralized Service		OpenDHT		Vuze (a.k.a. Azureus)	
	Satisfies?	Reason	Satisfies?	Reason	Satisfies?	Reason
Resistance to IP log collection & correlation:						
- by storage nodes	X		~-> X	Centralized OpenDHT management may log Ips	✓	Huge-scale, distribution protect against logging
- by bootstrap service	X	Service may keep & correlate IP logs	~-> X	Bootstrap node may log	~	Bootstrap node may log
- by infrastructure	X		~-> X	PlanetLab may log	✓	No infrastructure under Vuze
Resistance to overwriting attacks by clients	✓		✓	Password-protected mappings	~	No password, but only originator can overwrite
Scalability and scale	~	Normally, 10s-100s	~	Hundreds of nodes	✓	Millions of nodes
Configurable timeouts	✓		✓	Timeouts up to 1 week	X	8-hour timeouts too short
Good availability & persistence	✓		✓ -> X	Crashes hurt availability	~	Good persistence, but routing inconsistencies hurt availability
Stability and maintenance	✓		✓ -> X	Research prototype	✓	Production system
Backward compatibility	✓		✓	Designed as a service for apps	~	Vuze designed for one app.; compatibility not mandatory

Figure 1: **Comparison of Storage Solutions for Adeona (2008).** This table captures our perceptions of the storage solutions at the time in which we designed Adeona. For each requirement, a check mark means the storage solution satisfies the requirement, an "X" means it does not, and a twiddle means it only satisfies it in part. When our perceptions and understandings have changed due to our experiences, we list both our initial perceptions and our revised understandings separated by a right arrow.

Figures 1 and 2 respectively capture our views of the possible storage solutions for Adeona and Vanish when we designed these prototypes. These tables also capture our revised assessment of the suitability of these different storage systems based on our experiences. Because Vanish was designed after Adeona was deployed, the OpenDHT cells in Figure 2 are informed by our earlier experiences with Adeona and OpenDHT.

3.1 Experiences with Adeona and OpenDHT

As described in Section 2.1, Adeona was designed as a privacy-respecting method to track lost or stolen mobile device. Adeona's privacy goal means that the third party storage service should not be able to track a user's location before the device goes missing. Adeona uses cryptography to ensure that the *data* stored by the storage service does not reveal private information about a user's past locations. However, cryptography alone does not fully address Adeona's privacy goals.

When designing Adeona we were concerned about the possibility of the storage service profiling users based on *side channel information*. Specifically, even though clients' location data is encrypted, the storage service will still see the IP addresses of clients. The service could use knowledge of these IP address to profile users, determine when they leave for work and return home, and so on. The storage service could also use this side channel information to infer how many devices are behind a private network. Revealing this information to third party storage services is contrary to our privacy-preserving device tracking goals.

One approach for minimizing such profiling is to distribute the actual storage of information over a large collection of decentralized nodes. In a large decentralized system, no single node would be able to obtain enough side channel information to profile users. This led to our decision to store encrypted client locations in a DHT. There is a significant difference between evaluating a proposed concept on paper or in a simulated environment, however, and deploying, measuring, and learning from a real system. We therefore decided to build our research prototype on top of a deployed and maintained DHT. We considered two such DHTs: OpenDHT and Vuze (then called Azureus).

We had a number of reasons for selecting OpenDHT over Vuze. OpenDHT supported timeouts up to 1-week, whereas Vuze only supported timeouts of 8-hours. For a system designed to track lost or stolen laptops, 8 hours were clearly insufficient, and even 1-week seemed short. This strongly

influenced our selection of OpenDHT. Further influencing our decision was OpenDHT's public commitment to support research and new applications, and hence our belief that future changes to OpenDHT would not break backward compatibility with our Adeona prototype.

Still, we knew of a few potential weaknesses with OpenDHT. For example, we knew that OpenDHT ran on top of PlanetLab, and hence PlanetLab could potentially monitor the operations of OpenDHT, and hence also monitor and track Adeona users. We also knew that the OpenDHT maintainer – if he wanted to – could instrument OpenDHT to record IP addresses and hence profile Adeona users. For our research prototype, these did not seem like fundamental issues. Our goal was to understand what was possible from a research perspective, not build a product. We wanted to know whether a privacy-respecting device tracking system could be built on top of a DHT, and hence these issues with OpenDHT did not seem critical.

In support of our research goals, to elicit feedback, and encourage further research and development of privacy-respecting device tracking systems, we released our Adeona prototype as an open source application. We were surprised to find a significant number of real users downloading our prototype (over 100,000 downloads). The fact that many people might be using our prototype made us step back and reconsider our choice of OpenDHT as the underlying storage infrastructure. From a research perspective, the fact that PlanetLab can monitor all OpenDHT communications is not a significant issue (one could move OpenDHT to another underlying infrastructure, for example). Similarly for the fact that the OpenDHT maintainer can monitor all OpenDHT communications. Furthermore, from a practical perspective, we believe that the PlanetLab maintainers and ourselves (as the present maintainers of OpenDHT) are trustworthy and hence we would not profile users even if we had the capability to do so. Nevertheless, it is counter to the goals of Adeona for users to have to trust any of us. Hence, although OpenDHT was a logical decision for a research prototype, it is not the ideal platform for supporting Adeona in an environment with real users.

Separate from security, the most glaring problem we encountered after deploying Adeona was reliability. In the months following Adeona's public release, we started to slowly see the degradation in performance of OpenDHT, and OpenDHT would go for long stretches of time without having a single active node. Earlier this year (2009) the maintainer and original author of OpenDHT formally announced his intention to no longer support OpenDHT. This situation was problematic for us. Although Adeona is a research prototype, we felt an obligation to provided continued service for users who have downloaded the Adeona prototype, and therefore took over the maintenance of OpenDHT.

3.2 Experiences with Vanish and the Vuze DHT

As noted in Section 2.2, Vanish set out to meet a set of challenging goals for self-deleting data. Those goals led us in certain directions for the prototype. We ruled out a centralized service from the beginning, because we did not want to rely on any single trusted entity. In part, this was a result of the history of Hushmail [21], a secure email service that was found to be passing cleartext versions of its secure emails to the Canadian government. More important (and difficult), we wanted data to disappear "on its own," i.e., without any explicit action on the part of anyone. Therefore, we needed a means of computing or storing keys in which those keys would be lost through a natural process. We initially thought of two potential mechanisms, one based on the constant evolution of content on the Web and the second based on natural churn and data purging in a DHT. While both seemed interesting, we didn't feel we could simultaneously pursue both directions in any reasonable period of time, so we decided to focus on the DHT approach in our study and leave other methods for later work.

Having built and deployed the Adeona prototype previously, we were well aware of OpenDHT

VANISH Primary goal: data privacy						
Storage System Requirements	**Storage solution**					
	Centralized Service		**OpenDHT**		**Vuze (a.k.a. Azureus)**	
	Satisfies?	**Reason**	**Satisfies?**	**Reason**	**Satisfies?**	**Reason**
No single point of trust, control, and data visibility	X	Provider sees all and must be trusted	~	Conceptually decentralized, but must trust OpenDHT management and PlanetLab	✓	Completely decentralized (may bootstrap from node participating in torrents or cached information)
Huge scale	~	Normally, 10s-100s	~	Hundreds of nodes	✓	Millions of nodes
Geographical distribution	~	Usually, not so much	✓	PlanetLab-wide	✓	World-wide (~190 countries)
Churn, untraceable natural evolution	X	May keep traces of system structure	~	Little churn	~->X	DHT location changes w/ IP/port, but attackers may track DHT membership
Resistance to data harvesting:						
- Resistance to instrumentation	X	Can instrument service for data collection	X	OpenDHT management & PlanetLab can instrument	~	Vuze controls code, but code is open and auditable
- Resistance to crawling / sniffing	✓	Closed access	X	Closed access, but malicious PlanetLab nodes can learn joining key	~->X	Open access + over-replication enable efficient crawling; in general, not designed for privacy
- Resistance to hijacking	✓	Closed access	X	Closed access, but malicious PlanetLab nodes can learn joining key	X	Open access enables blackhole attacks
Configurable timeouts	✓		✓	Timeouts up to 1 week	X->✓	8-hour forcible timeouts
Good availability	✓		X	Crashes hurt availability	~	Good persistence, but routing inconsistencies hurt availability
Good performance	✓		~		X -> ~	Vuze not designed or tuned for time-sensitive applications
Backward compatibility	✓		✓	Designed as a service for many apps	~	Vuze designed for one app.; compatibility not mandatory
Stability and maintenance	✓		X	Research prototype	✓	Production system

Figure 2: **Comparison of Storage Solutions for Vanish (2009).** This table captures our perceptions of the storage solutions at the time in which we designed Vanish. When our perceptions and understandings have changed due to our experiences, we list both our initial perceptions and our revised understandings separated by a right arrow. *Note that our initial perceptions of OpenDHT for Vanish were affected by our experiences and reflections on using OpenDHT for Adeona. Namely, our OpenDHT column is more informed than in Figure 1.*

and its strengths and weaknesses. Our perspectives on OpenDHT when we designed Vanish were thus more well informed than when we designed Adeona. For Vanish, OpenDHT seemed a poor fit for a number of reasons. First, OpenDHT is a relatively small system (hundreds of nodes) and there is little churn. We were also experiencing reliability/availability problems with Adeona at the time, and there was the risk that the system would be shut down by its operator (which, as we noted in Section 3.1, did happen). Finally, while OpenDHT is (curiously) a closed system, requiring a password to gain admission, it is possible for a malicious PlanetLab node to determine the key and hijack the DHT. Overall, as a small-scale, centrally controlled research prototype, OpenDHT did not seem like a good fit for a Vanish proof-of-concept prototype.

Vuze, on the other hand, had a number of very promising characteristics. First and foremost, it was a production system of enormous size (over 1M simultaneously connected nodes [8]). Second, those nodes were scattered over nearly 200 countries [25], which makes the system difficult to attack from a legal perspective (one of our goals as well).

One issue for us with Vuze was its 8-hour timeout, which we knew was not really long enough for our application. By comparison, OpenDHT had configurable timeouts up to a week, but as noted above, it is other deficits ruled it out. Since we were building a proof-of-concept prototype, we decided to accept the 8-hour timeout, although we implemented a manual refresh service on the client side that could extend the timeout by re-pushing the keys. We were very pleased when shortly after we published the Vanish work, Vuze modified their system explicitly for Vanish, extending timeouts up to three days in 8-hour intervals.

In retrospect, the most significant issues were in the area of resistance to data harvesting. In part, we were counting on the astronomical DHT address space (160 bits in the case of Vuze) and large number of nodes (millions in the case of Vuze) to help us to "hide" data. In our evaluation, we implemented and measured a number of injection attacks where we inserted thousands of malicious

nodes into the DHT, but we made several assumptions about how expensive that attack would be based on the cost of the Vuze client. We had considered other approaches as well but were concerned that implementing them would violate the Vuze use agreement, which we did not want to do. Of course, a real attacker would not care about violating the agreement! Since that time, other groups have now devised attacks using extremely efficient clients that can quickly scan Vuze and capture its contents [31]. These attacks are successful both because of the attacker efficiency but also because of several aspects of the Vuze design.

The crucial property that we overestimated was the resistance of Vuze to crawling attacks. Basically, the Vuze design is focused on ensuring the persistence and availability of values in the face of significant churn in the DHT and of inconsistent routing tables [8]. As a result, replication mechanisms and levels in Vuze are designed to guarantee persistence and availability; they ignore information dissipation. We have identified two mechanisms in Vuze that make it particularly vulnerable to crawling attacks, though only one of them is exercised today by these attacks. The first is a push-on-join mechanism; when a new node joins the DHT, its neighbors quickly push copies of all of their content to the new node. This means that a malicious node that joins will see a huge benefit (obtain a lot of data) in a very short period of time (a few minutes). The result makes hopping attacks – where malicious nodes quickly move from place to place in the DHT address space – extremely effective.

The second mechanism is an (overly) active replication policy that causes every node to replicate each of its values to its 19 closest neighbors every 30 minutes, unless the value was pushed or replicated by its originator or by another replica within the past 30 minutes. Naturally, the vulnerabilities introduced by this periodic 20-way replication are never exercised by today's DHT crawlers, as an attacker never needs to wait in one place for more than a few minutes to receive the push-on-join. However, conceptually, if push-on-join did not exist, the 20-way, every-30-minutes replication would make Vuze still vulnerable to the crawling attack. As we will show later, this level of replication is unnecessary to maintain persistent values and is a gaping security loophole when supporting privacy applications.

The persistence/availability that Vuze provides was important to Vanish however: we wanted keys to be available with high probability until the data timeout, and then to disappear with high probability at the timeout or shortly after. We used Shamir secret sharing in Vanish for several reasons. First, even with the over replication in Vuze, we needed to ensure that the data would endure if the node storing a key crashes or leaves the system shortly after we push the key. Second, we used secret sharing to resist attack; an attacker would have to compromise many nodes (the "threshold" in the secret sharing parameter) to obtain our full key. Of course, if you could sniff the entire DHT then secret sharing would not help very much. By increasing the threshold (the number of key shares required to reconstruct the full key) we can increase resistance to sniffing, however there is a performance tradeoff in doing that.

Another property that we overestimated was Vuze's ability to withstand DHT infiltration attacks where an attacker hosts hundreds or thousands of nodes on a single machine. Interestingly, the Vuze designers had placed code in Vuze that would limit the number of nodes that could join from a single IP address. This code has the potential to reduce the impact of a Sybil attack that emulates thousands of nodes on a single physical node. The code was added to Vuze originally as a defense against anticipated denial-of-service attacks, but since those attacks never materialized, the code was never enabled. Enabling the code does have a potential downside of reducing the number of clients that can sit behind a NAT; we examine this in the next section.

In summary, we chose Vuze because of its pervasive use and physical properties; however the system was never intended to support a security application such as Vanish, and in the end it has significant security holes for such applications. Those holes were not important for Vuze applications

in the past, but if applications such as Adeona or Vanish will become commonplace, then Vuze or future DHTs will need to consider the needs of security and privacy applications in their design and implementation. We now discuss ways to do this.

4 Strengthening DHTs for Security Applications

The previous section described our experience and perspective with DHTs for security and privacy applications, and in particular, the conflict between the many attractive properties of DHTs ("the concept") and the details of the existing DHT implementations we chose ("the reality"). This conflict raises a difficult question: are DHTs totally inappropriate for such applications?

We cannot answer this question definitively here: that effort will require a large amount of future research. However, we hope to motivate that research by describing desired requirements for future DHTs intended to support security applications. We also provide some simple examples of how existing DHTs can raise the bar and improve their resistance to certain kinds of attacks.

4.1 New Requirements for DHTs for Security Applications

Security analysis of DHTs is not new, however, we believe that applications such as Adeona and Vanish require a fresh perspective. For example, a significant amount of prior research on DHT security has focused on *byzantine attacks*, *e.g.*, denial of service attacks where malicious nodes misdirect lookups, pollute routing table entries, hijack keyspaces, corrupt storage, and so on [1, 5, 9, 14, 11, 16, 19, 22, 23, 24]. In general, these are all attacks aimed to disrupt the DHT's functioning by degrading its performance, availability, or integrity.

In contrast, the greatest concerns for Adeona, Vanish, and possibly other future security applications are *information harvesting attacks*. For example, Adeona's primary goal is to preserve location privacy of its users; if a small number of DHT nodes were able to capture most of Adeona users' requests to the DHT, then they could log the locations from which those requests were made, thereby compromising Adeona's location privacy. Similarly, Vanish's primary goal is to preserve data privacy, but if an attacker could capture most of the Vanish key shares, then he can persist these shares past their timeout, thereby violating Vanish's privacy-via-self-destruction property.

One form of information harvesting mechanism that has been developed in the past is DHT crawling [26, 27, 28]. Previous measurement studies of DHTs have demonstrated that crawling a large-scale DHT like Vuze is possible with modest resources. However, these efforts had focused mostly on crawling *nodes*, *e.g.*, to derive properties of their geographical distribution, churn, and network structure. In contrast, security applications are concerned about crawling of DHT *content*. Conceptually, crawling content should be difficult because of the astronomically large key space (*e.g.*, 160 bits in Vuze); querying the key space of a node should be impractical for an attacker. However, as we've already discussed, the eager replication mechanism in a system like Vuze makes this key space size irrelevant.

Given this different perspective, future DHTs must include strengthened resistance to new attacks if they are to support security and privacy applications. In particular, we believe that they must include (at least) three principal forms of resistance, in addition to traditional DHT security goals:

1. *Resistance to data harvesting:* A DHT node can obtain only a small bounded amount of data if its tenure in the DHT is short.

2. *Resistance to pervasive infiltration:* A node cannot assume multiple positions inside the DHT at a given point in time.

3. *Resistance to intercepting requests:* A DHT node can eavesdrop on only a bounded number of updates and retrieves from clients if it has a short tenure.

Some of the above requirements may not seem entirely new. In particular, the need to resist pervasive infiltration was formulated in the context of the Sybil attack, while the need to limit request interception reminds one of the blackhole attack. These are both long-studied attacks with well-understood defenses. However, as we argue later in this section and in Section 6, most of the known measures are either too heavyweight or complex for today's DHTs to adopt or are not applicable for the privacy goals of our applications. For example, one way in which blackholes have been deflected in the past is to increase redundancy of lookup routes or data. However, including such a measure would result in revealing even more location information and data than not including it.

While no single defense (or even small set) provides perfect security, strengthening a DHT's resistance in these ways raises the defensive bar against an adversary attempting to obtain significant portions of the index-value pairs. In particular, we wish to make information harvesting attacks more difficult to scale by requiring the adversary to invest both *more time* and *more resources*. In the following subsections we provide some examples of increasing attack resistance in current DHTs. A natural approach for improving OpenDHT would be to transition it from a single-maintainer closed system to a restricted membership system. Vuze, on the other hand, is currently an open system, and hence it poses greater challenges. We therefore focus on Vuze.

4.2 Resistance to Data Harvesting

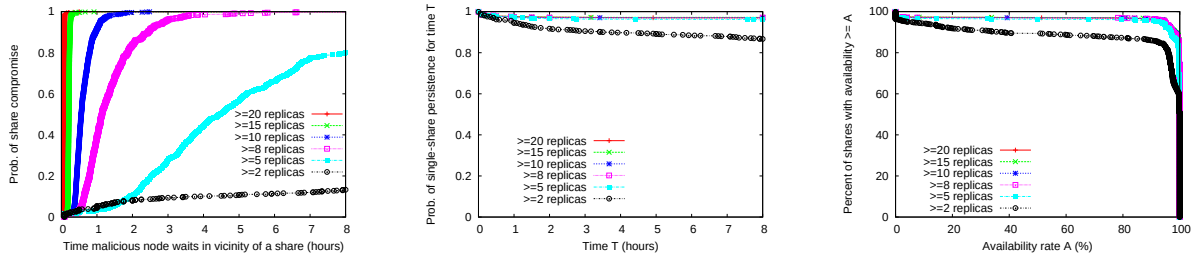
In this section we consider resistance to data harvesting in Vuze. Our goal is not to make Vuze 100% secure but to demonstrate that relatively simple changes in its current design — changes that can be easily implemented — can significantly increase resistance to these attacks.

Previously we described the Vuze replication policies: push-on-join (nodes immediately push data to a newly joining node) and periodic replication (one replica pushes data to 19 others once every 30 minutes). These policies make Vuze particularly vulnerable to DHT crawling attacks. For example, one particularly effective attack strategy — described in [31] — is to join the DHT with a set of nodes, wait just long enough to capture the data offered by other nodes in its vicinity, and then "jump" to a different set of locations.[1] We thus propose and evaluate simple changes to these two replication policies below.

Push-On-Join. The first defense is to simply disable Vuze's on-join replication mechanism, which fuels crawling attacks in the current Vuze system. We implemented this trivial change to be backward-compatible: a flag specified when a value is pushed tells the storing nodes whether or not to replicate that value to new nodes that join. However, by itself this defense is insufficient protection against the crawling attack; it only increases the attacker's required waiting time from a few minutes (the usual bootstrap time in Vuze) to close to 30 minutes, when Vuze's 30-minute replication kicks in. Still the enforcement of this defense is required for the second replication policy change to take effect.

Threshold Replication. Currently in Vuze, each of the 20 nodes closest to a data entry (key-value pair) replicates that data once every 30 minutes, unless it has already been replicated by

[1]The DHT ID of a node in Vuze is a function of its IP address and listening port, so changing the port number will change its DHT location.



(a) Prob. attacker obtains share via replication.

(b) Single-share persistence.

(c) Single-share availability.

Figure 3: **Evaluation of the threshold replication defense against DHT crawling attacks.** Fig. (a): Probability that one attacker node compromises a given share, if it becomes one the share's 20 replicas and waits for a time period for its replication; various replica thresholds are shown (*e.g.*, ">= 5" means a threshold of 5 replicas). Performing replication at a minimum threshold of 5 replicas requires an attacker to wait for ≈4h for maximum yield. Fig. (b) and (c): Persistence and availability properties of various minimum-replication policies; redundancy via secret sharing (*i.e.*, requiring 45/50 shares) can fully compensate for premature losses for 5-20 minimum replicas.

others within the past 30 minutes. This means that if an attacker node waits in one spot for 30 minutes, it can be nearly sure to capture all of the values in that spot.

A simple improvement is to replace this eager replication with a more conservative policy that replicates only when needed. That is, instead of blindly replicating data every 30 minutes, which risks its dissemination to sniffers, replication occurs only when the number of remaining replicas dips below a threshold. A client only needs one replica to retrieve the data, and if there are several replicas and little churn, then there is little need for further replication.

We call this policy *threshold replication* and its key parameter (the *threshold*) is the minimum number of replicas that nodes strive to maintain. By tuning this parameter we can strike a balance between security and persistence/availability. To evaluate this policy we modified Vuze so that a node checks with its 19 neighbors frequently (*e.g.*, every 5 minutes) to see how many copies of a table entry exist.[2] As long as the number of replicas remains above the threshold (*e.g.*, 5 replicas), no replication occurs. When the number of replicas dips below the threshold, the first node to notice sends the value to the other 19 nodes, restoring the replica set to 20 replicas. This policy is easy to implement (50 lines of code in Vuze and 5 lines in Vanish). Once again, a flag set by the application at store time specifies whether or not the value should be replicated with the thresholds scheme and provides the threshold to use.

Evaluation of Modified Replication Policies in Vuze. We designed an experiment to evaluate our policy changes (disabling push-on-join and adding threshold replication). In the experiment we join the Vuze DHT with 1,000 nodes for 20 hours. Each node performs two types of lookups. First, a node performs a lookup of its own ID once every 5 minutes to obtain a local view of the replica set for its own index (for which it is surely responsible at any time). Second, each node performs a lookup of the IDs of 20 other nodes in the experiment once every 30 minutes, in effect simulating remote clients that try to access some data at the nodes' indices. For each lookup, we record the returned set of nodes. Thus, the 1,000 nodes give us local and remote visibility into what would have happened had we pushed 1,000 values at the nodes' indices. Based on the replica set traces, we are able to compute replication intervals, persistence, and availability for the simulated

[2]To ensure that periodic checks reveal no information about populated table indexes, we have added a new function to the Vuze interface, which, given a hash, checks whether any entry exists at an index with that hash.

values, under various minimum replication thresholds.

Figure 3(a) shows, for different threshold values, the probability that an attacker who ends up among the 20 nearest nodes of a DHT index will capture the contents of that index, provided that it waits for a given period of time (shown on the x axis). The figure shows that actively maintaining a threshold of 20 replicas (the >= 20 line at the far left) requires replicating once every few minutes; this is even more aggressive than Vuze's default 30-minute replication. Hence, in this case, the adversary only needs to wait for a few minutes in order to have almost 100% chance to have seen a replenishing cycle. However, replenishing 20 replicas only when 5 replicas are left (the ">= 5" line) significantly increases the interval between replications; in this case an attacker must remain in its spot for about 3.8 hours for maximum yield.[3] Even for such long waiting times, the probability of capturing a *given* share is relatively low (0.447).

An important question is whether this policy sacrifices persistence or availability for increased security. Figures 3(b) and 3(c) show the persistence of DHT values over time for different thresholds. For a threshold of only 2 there is some loss of persistence. However, at a threshold of 5 there is little loss compared to a threshold of 20. And in the case of Vanish, any small losses in persistence and availability are mediated through the use of secret sharing, which tolerates a small number of lost key shares. This confirms our assumption that Vuze's replication policy is significantly over-engineered and over-provisioned for the current churn conditions. Similar conclusions can be derived from the results in a two-year-old study [8], which may indicate that over-replication has existed for a long time.

The results from Figure 3(a) allow us to compute the minimum number of nodes that an attacker needs to capture 25% of the Vanish-encrypted message expiring in 8 hours, provided that it remains in one spot for a period of time that maximizes its yield. For this computation we expanded a probabilistic model initially developed by the developers of the crawling attack [31] (see details in Section B). Our model shows that for Vanish, the minimum number of attacker nodes for compromising vanishing messages (with secret sharing parameters requiring 45 out of with 50 shares to reconstruct the key) are: ≈ 600 nodes for original Vuze (with 3-minute hops); $\approx 8,000$ nodes for threshold >= 10 replicas, $\approx 20,000$ nodes for threshold >= 8 replicas, and approaching $\approx 90,000$ nodes for threshold >= 5 replicas. Therefore, the difference between the current, overly eager Vuze replication scheme and threshold replication at 5 replicas is three orders of magnitude in terms of the number of nodes required by the attacker.

Overall, it's clear that a set of extremely simple and backward-compatible changes to Vuze can significantly raise the bar against a crawling attacker. This does not mean, however, that the technique is secure. Indeed, the threshold replication is vulnerable to clustering attacks, where an attacker concentrates many nodes (*e.g.*, 19 nodes) into one spot, thus tricking the remaining nodes to replicate. Similarly, $90,000$ nodes is not a large number for an attacker with an extremely lightweight infrastructure such as [31]. This requires additional resistance, which we now describe.

4.3 Resistance to Pervasive Infiltration

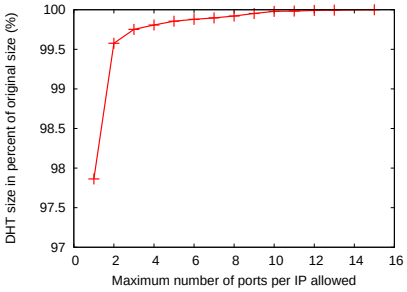
New DHTs should limit their vulnerability to large-scale infiltration by a small number of physical machines. We believe that in this case as well, a simple approach can have significant payoff.

At first glance, infiltration as described above appears to be the traditional Sybil attack problem. Unfortunately, the solution to traditional Sybil attacks requires a logically centralized authority that assigns strong and persistent identities to participating peers in order to monitor them [7]. From a practical perspective, we do not expect the adoption of strong identities in a P2P DHT like Vuze.

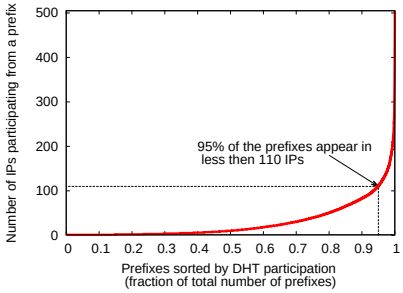
[3] It can be shown that the maximum yield for the attacker is achieved at the point that maximizes the ratio of the capture probability to the time spent in the DHT; essentially, we find the knee of the curve.

Clients/IP	Number Observed
1	426368
2	5052
3	355
4	106
5	52
6	28
7	21
8	11
9	12
10	14
10–20	21
20+	10

(a) Measurements of clients/IP over 8 hours.



(b) Restricting number ports per IP.



(c) Restricting number of IPs per BGP prefix.

Figure 4: **Strategies to Resist Pervasive Infiltration.** (a) Measurements at the Vuze bootstrap server regarding number of repeat IPs seen. (b) Impact of restricting the number of ports per IP on DHT size in Vuze; restricting the number of ports to 10 results in less than 1% loss in DHT size. (c) Distribution of BGP prefixes in Vuze; restricting the number of IPs per BGP prefix to 505 almost guarantees that no legitimate user will be ousted.

On closer examination, however, the infiltration problem for our applications is more tractable than a Sybil attack. Consider the use of a centralized bootstrapping service that uses IP addresses for node identities and prevents multiple DHT joins by nodes bearing the same IP. This technique is not an effective solution for the Sybil attack problem in a general P2P system, as it would mean that two clients behind a NAT cannot simultaneously participate in the P2P system. A complete solution to the Sybil attack problem has to both: (a) deny Sybil nodes admission, and (b) ensure that any legitimate node can participate.

In the context of a DHT service, however, we believe that solutions that provide (a) but not necessarily (b) are still viable. Denying admission to clients with duplicate IPs will reduce the number of nodes participating in the DHT (*i.e.*, nodes implementing DHT storage). However, nodes that are denied admission in the DHT can still operate as DHT *clients* (*i.e.*, they can perform stores and lookups). Such nodes can thus participate in BitTorrent swarms, publish in Vanish, and so on, yet be unable to easily use DHT mechanisms (such as replication and visibility of storage operations) for information harvesting. IP addresses can therefore serve as weak identities that can improve the security of the DHT for applications like Vanish without limiting its usability.

Refining Infiltration Resistance. We can create a more stringent requirement for infiltration resistance based on using IP addresses as node identifiers, and can also consider malicious attacks perpetrated by stronger adversaries who can control an entire BGP prefix. Our new requirements are: (a) at most a few nodes with a certain IP can be participating in the DHT at a given point in time, and (b) a single BGP prefix cannot have a disproportionately high representation in the DHT. The second requirement is to prevent infiltration by an adversary who owns a large chunk of the IP address space, such as a */8* prefix. We show that both requirements can be met by making a few simple modifications to how node IDs are generated in the DHT.

In Vuze and many other DHTs, a joining node's ID is generated by computing the SHA1 hash of the node's publicly visible IP address and port number, *i.e.*, *H(n) = SHA1(IP(n), Port(n))*. With such a scheme, a node can join the DHT at multiple locations by communicating concurrently over different ports. To limit this attack, we can restrict the number of IDs that a given node can utilize to k by using the modified hash function, *H(n) = SHA1(IP(n), Port(n) % k)*. This limits the number of DHT locations that an adversary can infiltrate, as well as limits clustering attacks by tightly binding DHT locations to IPs. But this defense would also reduce the size of

the DHT as multiple nodes behind a given NAT cannot simultaneously participate in the DHT. We quantify this reduction using measurements gathered at the Vuze bootstrap server over an eight hour period (see Table 4(a)). These measurements show that most clients have unique IP addresses and limiting the number of clients per IP address results in a small reduction in DHT size (see Figure 4(b)). Thus, admission control restrictions will have only a marginal impact on DHT performance, while significantly toughening the DHT to withstand infiltration attacks. This defense is already implemented in Vuze, but is not enabled because attacks exploiting this property had not previously materialized.

The second requirement can also be realized by modifying the node ID generation function. As with ports, we can restrict the number of IPs that are active within a single BGP prefix. We first decompose the IP address into two components, its BGP prefix and its offset within the prefix. We then compute the node ID as *H(n) = SHA1(Prefix(n), Offset(n) % p, Port(n) % k)*, so as to limit the number of participants from a single prefix to at most $p * k$. Each DHT client can periodically download the IP to prefix mappings from BGP repositories such as Route Views and RIPE. This is an effective strategy given that Vuze clients are typically scattered across many BGP prefixes, with most prefixes having fewer than a few hundred representatives at a given point in time (see Figure 4(c)). Since the number of nodes required to harvest the DHT is in the order of tens of thousands of nodes, massive infiltration from a small number of prefixes can be easily detected. While not completely preventing infiltration, these two strategies would raise the bar against DHT-global infiltration attacks.

4.4 Resistance to Intercepting Requests

We have suggested simple point solutions to Vuze to prevent data harvesting attacks. We now consider techniques that would further harden DHTs by reducing the communication they would see as well as the the number of values they would store. While the proposal here is not supported by experimental analyses or measurements, and is hence more speculative than our proposals above, we believe that this technique is viable and provide a more detailed description in Appendix A.

The primary principle underlying our proposed defense is to limit the keyrange allocated to a node based on its perceived age in the system. Nodes are initially assigned only a small portion (δ) of the keyspace. The allocated keyrange doubles exponentially over time, but the exponential doubling stops when the key range is as big as that of its predecessor. Since most queries are routed through the predecessor in a traditional DHT, the predecessor serves as the gatekeeper and can exercise effective control over what keyrange is handed over to the new node. Further, each node maintains a local estimate of a neighbor's age and is therefore resistant to various forms of Sybil and disinformation attacks.

There are a number of pleasing qualities to the scheme outlined above. As a greater portion of the keys are assigned to only nodes that have been participating in the DHT for a while, the scheme supports several of our resistance properties (resistance to data harvesting and intercepting communication). The latter is a consequence of the property that the lookup traffic conveyed by a DHT node is proportional to its allocated keyrange. Further, since long-lived nodes are typically stable, the proposed key allocation scheme provides greater levels of robustness and availability in addition to improving security.

4.5 Summary

This section presented several approaches to hardening DHTs against data harvesting attacks. Many of these techniques could be easily added to today's DHTs, which were not designed for

security and privacy applications. For new DHTs designed in the future, however, we believe that the types of resistance we defined in Section 4.1 should be included from the start.

While these defenses will raise the bar against attackers, requiring them to use more resources, today's DHTs are unlikely to be 100% secure. However, this does not necessarily imply that DHTs cannot play a role in supporting security applications. In the next section we show how applications built on top of DHTs can increase their own security and privacy.

5 Strengthening Security Applications via Architectural Changes

Previous sections have described our experience with Adeona on OpenDHT and Vanish on Vuze, a number of DHT weaknesses that leave these systems open to security attacks, and several potential approaches to hardening DHTs against those attacks. We believe that new DHTs in the future can be more resistant to attack or can at least raise the bar for attackers. However, given that DHTs are in general intended to be open structures shared by non-trusting clients, it's not yet clear how secure such structures will be in the end. Needless to say, much work remains to be done.

From our analysis above, though, it is clear that different DHTs have different strengths and weaknesses with respect to the needs of security and privacy applications. For example, OpenDHT is relatively small but more tightly managed; Vuze is enormous but unmanaged. OpenDHT protects mappings against unauthorized writes; Vuze only does so partially. This leads to an obvious question: can we harness the features of *multiple DHTs* in order to strengthen the security properties of our DHT-based applications, achieving the union of the best properties?

We believe that security-based applications can benefit from a *hybrid approach* in which they leverage the capabilities of multiple underlying systems, including both DHTs and centralized services, each with a different set of strengths and weaknesses. In the case of an application such as Vanish, employing the hybrid approach is straightforward. In fact, we mentioned it briefly in [10] but without the same emphasis or perspective. For example, Vanish uses Shamir secret sharing to distribute keys over Vuze nodes in such a way that no single node has enough information on its own to decrypt the data. The extension is to distribute Vanish key shares over *both* OpenDHT nodes and Vuze nodes, in such a way that keys from both systems would be required to decrypt a Vanish data object. As a result, an attacker would have to subvert both OpenDHT and Vuze to break a Vanish message. This scheme could be further extended to include a privately managed "trusted" service as well, or even several of them. Scattering keys over both highly distributed (and hopefully more attack-resistant) DHTs with different security properties, and centrally managed services, would make attacks extremely difficult and costly; each attack would require different strategies, possibly including physical attacks. Note that the DHTs still play an important role here, since the user does have have to trust any single centralized service, and the loss of sufficient keys in any DHT would cause data to self-destruct.

We have implemented a version of Vanish built on top of OpenDHT and Vuze as described above and will be distributing that code in the near future.

The above hybrid strategy, while suitable for Vanish, is not suitable for all security applications however. For example, for Adeona we seek to minimize the possibility of a service profiling users based on the locations from which the users connect to that service. Simultaneously sending Adeona location data to multiple DHTs would therefore actually weaken Adeona's privacy properties. While anonymity systems like Tor [29] could be used to hide the IP addresses of clients from storage services, we can leverage the hybrid approach for applications like Adeona in a different way. For Adeona, the hybrid approach would pseudorandomly time-multiplex client location updates across multiple DHTs and possibly also managed services. Time-multiplexing client updates across DHTs

would limit the side channel exposure of IP addresses to each individual DHT.

Finally, we remark that the traditional use of DHTs require a bootstrapping phase in which clients learn how to join or access the DHT. To avoid profiling by the bootstrapping process, applications like Adeona could use a client-side cache of recent DHT nodes and bootstrap locally.

6 Related Work

As mentioned before, there has been tremendous body of work on *byzantine attacks* on DHTs. For example, one of the first surveys of byzantine threats [23] introduces attacks such as blackholing routing tables, partitioning DHTs, corrupting data, etc. Other work provides quantitative studies of the impact of such attacks [24, 30], And yet other work introduces defenses against such purposes [5, 22, 14]. For example, [5] proposes methods for securing routing table maintenance and message forwarding in order to prevent blackhole and routing pollution attacks; [22] creates techniques to prevent a particular region of the keyspace from being hijacked by adversaries; and [14] discusses various resource allocation mechanisms to prevent DoS attacks from degrading DHT performance.

The common theme in these attacks and defenses is that they are aimed at disrupting the DHT's functioning by degrading its performance, availability, or integrity. Such attacks are indeed the most viable threats in traditional applications on DHTs — torrent tracking and P2P file systems [6] — which either use the DHT as a *publish* medium (*e.g.*, torrent information) or employ end-to-end cryptography to protect data privacy. While these attacks — and some of their defenses — apply to Vanish and Adeona, as well, these applications introduce a new set of attacks, prominently *information harvesting attacks*. This paper introduces them and presents a set of requirements that future DHTs must satisfy in order to prevent such attacks.

Some of these requirements may not seem new. In particular, the need to prevent pervasive infiltration was identified in the context of the Sybil attack [7], for which many defenses are known. For example, the original Sybil paper [7] postulates the need for strong identities to solve this problem; others propose the use of computational puzzles and bandwidth contributions to make peers prove that they are not Sybils [4]; and others suggest taking network structure into account to detect Sybils (SybilGuard [32]). Unfortunately, none of these defenses have been adopted by today's DHTs like Vuze, in part because no real need was perceived in the context of existing applications and in part because many of them were deemed too complex or heavyweight. We argue that simpler, more practical solutions would make a great difference with respect to infiltration attacks and propose an admission control mechanism that relies on IP addresses as weak identities and separates service nodes from client nodes. In our model, anyone can obtain service from a DHT (*i.e.*, can get or put values), but not everyone can participate in the DHT. This model is similar to that of OpenDHT [17], albeit its use there is motivated by different reasons: to create a DHT service that can support many applications rather than to increase security.

Some of the attacks may not seem new, either. Various measurement studies of DHTs have demonstrated that crawling the nodes in existing giant-scale DHTs is practical. For example, various measurement studies [28, 27, 26] have illustrated that crawling the KAD and Vuze DHTs to characterize the network properties All of these studies perform *nodes crawls*, *i.e.*, monitor DHT membership in order to characterize some aspect of the DHT network. Vanish, however, is vulnerable to *data crawls*, in which adversaries are interested in capturing all data in the DHT. Such crawls are fundamentally different, and, although proven practical by others [31] in the current Vuze, we believe they are also easier to defend against. Because information is "hidden" within a gigantic address space and because smarter replication techniques such as the ones proposed here can help limit data dissemination, we believe that data crawl attacks are not as fundamental as

node crawls.

More generally, this paper argues that designing DHTs for security applications such as Adeona or Vanish demands a fresh perspective, both in terms of the attacks and in terms of the defenses. For example, a range of defenses have been proposed elsewhere for DHTs that are at odds with the goals of Adeona and Vanish. In particular, certain proposals rely on increasing the redundancy level in both storage and routing to counter blackholing nodes [9, 16, 11]. Such solutions would in fact be detrimental for both Adeona and Vanish, as they would disseminate information (data in the case of Vanish and location information in the case of Adeona) even more than in the existing systems. In the paper, we give examples of the types of defenses that are suitable for our applications, although much future research remains to be done to strengthen DHTs for security applications.

7 Conclusions

This paper presented a retrospective on our experience with designing, prototyping, and deploying two recent DHT-based privacy prototypes, Adeona and Vanish. We discussed the new privacy demands placed on DHTs by applications such as these, the mismatch between these demands and those of traditional DHT applications, and the result in terms of the openness of existing DHTs to attack.

Examining the needs of security applications such as Adeona and Vanish and the strenghs and weaknesses of existing DHTs for supporting those needs provides useful insight, both for improving existing DHTs and designing future ones. We present a set of desired properties for increasing the resistance of DHTs to data harvesting and sniffing attacks. We also suggest several design changes — many easy to implement in Vuze in a backward-compatible way — that can raise the bar against such attacks.

Finally, we show how security and privacy applications can increase their security while using DHTs through a hybrid approach that allows them to benefit from the advantages of multiple types of systems, including DHTs and centralized services. We intend to distribute modified versions of our applications and of Vuze to experiment both with the hybrid approach and increased resistance to attack in DHTs

8 Acknowledgments

We are highly indebted to Alex Halderman and Scott Wolchok of University of Michigan and Ed Felten and Nadia Heninger of Princeton for discussing with us their deep insights into their clever Vuze sniffing attack. This attack was independently discovered by Brent Waters, Emmett Witchel, and colleagues at the University of Texas and resulted in [31]. This work was supported by NSF grants NSF-0846065, NSF-0627367, NSF-0614975, NSF-0619836, NSF-0720589, and NSF-0831540, an Alfred P. Sloan Research Fellowship, the Wissner-Slivka Chair, and a gift from Intel Corporation. This paper reflects the opinions and research of the authors, not necessarily those of their employers.

References

[1] B. Awerbuch and C. Scheideler. Group Spreading: A Protocol for Provably Secure Distributed Name Service. In *Proc. Int. Colloquium on Automata, Languages, and Programming*, 2004.

[2] Azureus, now called Vuze. `http://azureus.sourceforge.net/`.

[3] M. Bellare and B. Yee. Forward-Security in Privatey-Key Cryptography. In *Proc. of CT-RSA*, 2003.

[4] N. Borisov. Computational puzzles as Sybil defenses. In *Proc. of the Intl. Conference on Peer-to-Peer Computing*, 2006.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. of Symposium on Operating Systems Design and Implementation*, 2002.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of Symposium on Operating Systems Principles*, 2001.

[7] J. R. Douceur. The sybil attack. In First International Workshop on Peer-to-Peer Systems, 2002.

[8] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a million user DHT. In *Proc. of Internet Measurement Conference*, 2007.

[9] A. Fiat and J. Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Proc. of ACM Symposium on Discrete Algorithms*, 2002.

[10] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proc. of Usenix Security*, 2009.

[11] K. Hildrum and J. Kubiatowicz. Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-peer Networks. In *Proc. of International Symposium on Distributed Computing*, 2004.

[12] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring adoption of DHTs with OpenHash, a public DHT service. In *Proc. of IPTPS*, 2004.

[13] Lojack. http://www.absolute.com/products/lojack.

[14] P. Maniatis, T. Giuli, M. Roussopoulos, D. S. H. Rosenthal, and M. Baker. Impeding attrition attacks in P2P systems. In *Proc. of ACM SIGOPS European workshop*, 2004.

[15] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proc. of International Workshop on Peer-to-Peer Systems*, 2002.

[16] M. Naor and U. Wieder. A simple fault tolerant distributed hash table. In *Proc. of International Workshop on Peer-to-Peer Systems*, 2003.

[17] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In *Proc. of ACM SIGCOMM*, 2005.

[18] T. Ristenpart, G. Maganis, A. Krishnamurthy, and T. Kohno. Privacy-preserving location tracking of lost or stolen devices: Cryptographic techniques and replacing trusted third parties with dhts. In *Proc. of Usenix Security*, 2008.

[19] C. Scheideler. How to Spread Adversarial Nodes? Rotate! In *Proc. of ACM Symposium on Theory of Computing*, 2005.

[20] A. Shamir. How to share a secret. *Communications of the ACM*, 1979.

[21] R. Singel. Encrypted e-mail company Hushmail spills to feds. `http://blog.wired.com/27bstroke6/2007/11/encrypted-e-mai.html`, 2007.

[22] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *Proc. of ACM SIGOPS European workshop*, 2004.

[23] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. of International Workshop on Peer-to-Peer Systems*, 2002.

[24] M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *Proc. of Annual Computer Security Applications Conference*, 2004.

[25] M. Steiner and E. W. Biersack. Crawling Azureus. Technical Report RR-08-223, 2008.

[26] M. Steiner and E. W. Biersack. Where is my Peer? Evaluation of the Vivaldi Network Coordinate System in Azureus. In *Proc. of Networking*, 2009.

[27] M. Steiner, T. En-Najjary, and E. W. Biersack. A Global View of KAD. In *Proc. of Internet Measurement Conference*, 2007.

[28] D. Stutzbach and R. Rejaie. Understanding Churn in Peer-to-Peer Networks. In *Proc. of Internet Measurement Conference*, 2006.

[29] The Tor Project, Inc. Tor: Anonymity online. `http://www.torproject.org/`, 2007.

[30] A. Tran, N. Hopper, and Y. Kim. Hashing it out in public: Common failure modes of DHT-based anonymity schemes. In *Proc. of WPES*, 2009.

[31] S. Wolchok, O. S. Hofmann, E. W. Felten, J. A. Halderman, C. J. Rossbach, B. Waters, and E. Witchel. Defeating Vanish with low-cost Sybil attacks against large DHTs, 2009. Manuscript.

[32] H. Yu, M. Kaminsky, P. B. Gibbons, and A. D. Flaxman. SybilGuard: defending against sybil attacks via social networks. *ACM SIGCOMM*, 2006.

A Assigning keyranges based on age estimates

We first introduce the notation necessary to describe our technique for allocating keyranges based on a node's lifetime. Let $H(x)$ be the hash value of node x (say by hashing the IP and port number), and $PH(x)$ be the phony-hash value that it is assigned when it joins the DHT. Initially $PH(x)$ will be close to $H(x)$ but it could drift over time. Let $S(x)$ be the successor of node x in the DHT. The key range assigment works as follows.

- Each node x maintains the keys in the key range $PH(x)$ to $PH(S(x))$.
- When a node y joins the DHT it does a query for $H(y)$. Based on the lookup, it finds its immediate predecessor x and the immediate successor z. Note that x, y, and z would have to satisfy the property $PH(x) < H(y) < PH(z)$. In a traditional DHT, y will take over the keys in the key range between its value and its successor's hash value. In contrast, we do the following. $PH(y) = PH(z) - \delta$, where δ is a configurable and typically small value. y then stores the keys in $(PH(y) \ldots PH(z))$.
- Over time, we increase the keyrange allocated to a node. When x's estimate of y's age is $t_x(y)$ epochs, x will assign a larger keyrange to y such that $PH(y) = max((PH(x)+PH(z))/2, PH(z) - delta * 2^{t_x(y)})$. That is, the key range allocated to y is doubled as long as it does not exceeds half of the key range between x and z.
- When y leaves the system, x takes over y's key range. Note that x can do this because of the DHT routing property that requests to y's key range will typically go through x, especially when nodes realize that y is no longer alive. Further, now that x might potentially have a new successor, say r, it will start allocating portions of its key range based on how long it has known r. So, for instance, if y is replaced by r that is even younger from x's perspective, then its allocation will grow slower than y used to. That is, $PH(r) = max((PH(x) + PH(S(r)))/2, PH(S(r)) - delta * 2^{t_x(r)})$. Essentially x attributes a lower reputation to r than it used to do for y and trusts it less. If on the other hand, x's successor is a node that it has known for a while (such as its original successor z), it will do a load balancing operation such that x and z have similar number of keys.

A few points are worth noting. First, the outlined strategy improves overall system reliability since a node's current lifetime is highly correlated with its overall lifetime in a typical P2P setting. Second, it also improves load balance. In traditional DHTs, there could be a $O(log(n))$ load imbalance because nodes are assigned random node IDs. In our proposed extension, all long-lived

nodes end up having similarly sized key ranges because of the load balancing between a node and its successor. Third, the use of a $PH(.)$ value that is distinct from a node's hash is not a security concern. While the $PH(.)$ value could be arbitrarily different from its original $H(.)$ value over time, it does not enable nodes to pick arbitrary locations to join as long as current residents of the DHT perform verification of the inequality $PH(x) < H(y) < PH(z)$ at join time. Finally, sybil attacks within the same keyrange provide no additional benefit. For instance, if a sybil y' joins as the successor of y and y leaves the system, then y' will be worse off than y in its ability to harvest keyrange values off x as x either wouldn't have known y' or had heard about it just recently. If y' joins as the predecessor of y, then again it is provided only a small sliver and the cumulative size of the slivers handed out is not superlinear in the number of sybils used. So as long as the DHT implements the admission control policies, it remains robust.

B Probabilistic Model For Crawling Attackers

To model crawling attackers, we provide a probabilistic model that is an extension of the model developed by the authors of the crawling attack [31].

A crawling attacker joins with a set of A nodes and jumps around every t seconds. The attacker's goal is to capture as many Vanish-encrypted messages as possible via replication, assuming that shares are distributed at random indices in the DHT's space. We want to find the probability that the crawling attacker obtains a specific Vanish-encrypted message via replication before it expires (say in 8 hours). We define this probability in several steps:

P_1: Probability that one attacker node compromises a particular share of a Vanish-encrypted message if he waits for a time t in that share's vicinity. $P_1(t)$ is obtained directly from the experimental results displayed in Figure 3(a).

P_2: Probability that one attacker node ends up in a particular share's vicinity: $P_2 = \frac{20}{DHT}$, where DHT is the number of nodes in the distributed hashtable (typically over 1M).

P_3: Probability that one attacker node compromises a share if it jumps every t seconds, provided that the share times-out after 8 hours: $P_3(t) = 1 - (1 - P_1(t) * P_2)^{\frac{8h}{t}}$.

P_4: Probability that A attacker nodes compromise a share if they jump every t seconds: $P_4(t) = 1 - (1 - P_1(t) * P_2)^{\frac{8h}{t} \times A}$.

P_5: Probability that A attackers compromise a Vanish-encrypted message consisting of N shares, M of which are required in order to reconstruct the decryption key: $P_5(t) = \sum_{i=M}^{N} \binom{N}{i} * P_4^i * (1 - P_4)^{N-i}$.

The final probability, P_5, represents the probability associated with a successful crawling attack.