Adaptive Control for Anonymous Network-Attached Storage

Muthian Sivathanu, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau
Department of Computer Sciences, University of Wisconsin, Madison

Abstract

We describe techniques for building an anonymous network-attached storage system. Central to our design is adaptation, which enables higher performance without administrative tuning. Via a range of simulation experiments and a prototype implementation, we quantify the costs of providing anonymous storage and explore the parameters to which system performance is sensitive. We find that adding anonymity to a network-attached storage system is indeed feasible, increasing the network load by only a small, constant factor.

"Regular presentation", Student paper
Contact email: muthian@cs.wisc.edu
Mail: 1210 W Dayton St., Madison, WI 53706

1 Introduction

The manner in which data is stored and accessed is rapidly changing. In particular, network-attached storage devices (NASD) are an increasingly popular storage paradigm [8]. By placing each disk of the storage system directly on the network, NASD increases file system performance and availability, while lowering costs.

With these new potential benefits come new concrete problems. In the NASD paradigm, every file transaction is exposed on the network. In a typical file access, a client may communicate with a remote file manager to find the location of a particular disk block, and then communicate directly with the disk to obtain the desired data. No operations stay within the confines of a machine, since no machine has a disk attached directly to it.

Early research in protecting these operations has focused on ensuring the integrity [9] and privacy [10] of communication between clients, managers, and the disks themselves. With integrity provided by cryptographic checksums, users can be assured that an intruder cannot easily manufacture a packet and masquerade as someone else. With privacy provided by encryption, users can additionally feel confident that the exact contents of file system communication are not easily discerned.

However, guarantees of integrity and privacy are not sufficient to hide all aspects of file system activity from malicious entities. For example, an intruder can monitor which nodes are communicating with one another, and in doing so, potentially garner valuable information both as to which files a user is accessing and as to the structure of the file system itself. Such information may be useful in constructing attacks on systems that implement Byzantine fault tolerance [1]; if an attacker can control over one third of the nodes in such a system, secure file access can no longer be guaranteed. Also, it may be useful in constructing highly-focused attacks on the file system; for example, a denial of service that targets the file manager could be quite effective.

To achieve a higher level of security, anonymous protocols are required [2, 3, 4, 6, 11, 12, 13]. Anonymous protocols take additional measures beyond encryption in order to guarantee that an attacker can learn little or nothing about the nature of on-going communication. In this paper, we explore the addition of anonymity to network-attached file systems.

One of the most important issues in anonymous systems is that they often exact a high performance cost, due to the large amount of extra communication involved. While performance may be of less relevance in other domains, it is of utmost importance in a file system. Thus, we quantify the costs of a range of anonymity schemes, primarily focusing on the amount of additional network load.

We find that the key to minimizing the costs of anonymity is *adaptation*. We propose and evaluate a number of adaptive controllers that improve performance by adjusting the implementation of the anonymous scheme online, based on current usage patterns. By decreasing the amount of wasted data that must be sent to maintain anonymity, these controllers maximize performance, while ensuring that useful information is not exposed to attackers.

We demonstrate the efficacy of our schemes via a range of simulation experiments, and then confirm our results by implementing and measuring a restricted subset of the protocols in a network-attached file system prototype. Overall, we find that anonymity can be added to a network-attached storage system with reasonable cost. In the best case, only a factor of two network-load increase is incurred. We also observe that although throughput is reasonable, per-block latency is high, and therefore may limit the applicability of our anonymous techniques to applications that can tolerate latency. We find that adaptation is crucial in adjusting to the workload and system configuration without requiring human intervention. Finally, early experience with a prototype implementation indicates that our approach can be successfully realized.

The rest of this paper is structured as follows. In Section 2 we describe a basic NASD system, our threat model, and our approach to achieving adaptive anonymity. We present our simulation results in Section 3, our preliminary implementation results in Section 4, and related work in Section 5. Finally, we conclude in Section 6.

2 Design Alternatives

We now discuss how anonymity can be layered into a network-attached storage system. We first describe a NASD system, and then discuss our threat model. We then discuss our design goals, our approach for anonymity, and our adaptive controllers.

2.1 Network-attached Storage

We describe briefly the architecture of a network-attached storage system (or NASD system); a detailed description can be found in [8]. There are three components in the system: *clients*, one or more *disks*, and a *file manager*. Each communicates with one another over a general-purpose network.

A client is a general-purpose host in the system that accesses files through a file system API. A network-attached disk stores data on behalf of clients. Each disk also has the necessary processing capabilities to perform checksums or encryption of data, as well as sufficient meta-data to map and authorize requests to disk sectors [8]. The file manager is responsible for managing higher-level functions of the file system such as namespace operations (*e.g.*, the directory hierarchy) and access control. In terms of security, responsibility is spread across both the manager and the disks, with policy decisions made by the manager, but enforcement of those decisions performed at the disks themselves [8].

2.2 Threat Model

When providing anonymity, we consider the threat of both passive and active attackers. A passive attacker can only monitor communication, with the hope of learning something about the contents and nature of file traffic. In contrast to some previous work in wide-area anonymity schemes [13], in the local area one must assume that an attacker has access to all ongoing communication.

An active attacker (or "man in the middle") not only monitors existing communication, but may also manipulate communication to either cause harm or garner additional information about the storage system or its users. In particular, the active attacker may delay, drop, or replay messages.

Note that we assume that the endpoints of the system, including clients, disks, and the manager, are secure and uncompromised; if one or more of them are compromised, providing anonymity (or even privacy) of communication is likely to be of little value, since the attacker has access to many items in cleartext form (*e.g.*, in machine DRAM). However, our anonymous schemes assist the system in maintaining this high level of security by hiding information about the storage system from would-be attackers.

2.3 Design Goals

We next present our specific goals in designing an anonymous storage system. Our goals are threefold: in addition to the contents of each message, an anonymous storage system should hide the following information from the network attacker: (1) the *destination* disk of each client request (and conversely, the disk sending the reply), (2) the *identity* of the file manager, and (3) the *timing* of disk responses. We describe each of these in detail below.

First, by observing the amount of traffic destined for each disk or the timing relationship between client requests and arrivals at each disk, an attacker may be able to discern to which disk a client is sending requests. This knowledge can lead to two problems. First, the attacker may discern which disks contain valuable data, encouraging the attacker to focus on these disks in subsequent attacks while ignoring other disks. Second, if the allocation of files

across disks is known, then the attacker is given a hint as to the specific files the client is accessing.

Second, by looking for common access paradigms and message sizes, the attacker may learn which host is the file manager, because communication with the file manager tends to be in smaller message sizes. Also, certain patterns may be readily discerned when partaking in a security protocol or file lookup. The file manager is central to both the performance and security of the file system, and knowledge of which host is the manager enables the attacker to focus upon it. For example, a denial of service attack aimed at the manager likely cripples the entire file system, and gaining control of the manager gives an attacker access to various secrets stored therein.

A third piece of information that may leak out are the performance characteristics of the disk. An observer can monitor how long requests take to process and infer which type of drive is servicing the requests. Knowledge of drive type may enable an attacker to more easily gain access to the drive, as they may know which attacks will be successful upon that drive. Sophisticated disk fingerprinting tools are already available and may ease the attacker's task [15].

Note that in hiding this information from attackers, we have a slightly different goal from most anonymous communication systems [13]. In most systems, the idea is to protect the user of the system from an attacker who wishes to know with whom they are communicating. In our approach, anonymity is used primarily for increased security of the storage system itself, by hiding aspects of the storage architecture or data that may be useful in guiding subsequent attacks.

2.4 Anonymity

Before describing our approach that meets our goals for anonymity, we begin by describing the mechanisms required for a more extreme system that not only hides with which server a client is communicating, but also that the client is communicating at all. We then relax these constraints to improve performance while exposing little extra information. Note that privacy of communication is required and therefore implemented in our system. We provide details of how privacy is achieved in the appendix; our approach is similar to Miller *et al.*'s [10].

Throughout our discussion, we divide the network-attached storage system into two groups: the client group, which consists of the hosts that access file data, and the server group, which consists of the disks that serve data and the file manager. We refer to the group of servers as the "cloud of anonymity", as disk and manager activity are uniformly hidden outside of this grouping.

In the extreme approach for providing anonymity, each client constantly communicates with each server node and each server node constantly replies to each client. When a client or server has no real data to send, it must generate a *dummy message* to fill the void, which is indistinguishable from a real request to a network observer. A significant downside of this approach is that it consumes a large amount of network bandwidth, especially given a large number of clients with typically bursty workloads.

Thus, we investigate approaches in which the client only communicates when it has a valid request. In our approach, when a client has a request for data, it passes the request to a random intermediary server in the server cloud. Through a series of steps in which each server that receives the request forwards it to the next server, the request eventually arrives at the destination server which then handles the request. The exact route a message takes is determined by the *topology* of the server cloud; the topology also ensures that each disk forwards the same number of messages and is described in more detail below. After a request is handled, the reply moves through the server cloud back to the original server node, which passes the reply back to the client.

To provide anonymity at the level described in our goals, we discuss the requirements of the system:

• **Communication must be encrypted.** Cleartext would reveal which objects are being requested.

• **Servers must communicate at regular intervals.** If they did not, timing information would be leaked. Dummy messages should be used to fill intervals in which there are no valid requests. The interval at which the servers within the cloud communicate is known as the *forwarding interval*. Although uniform across servers, the interval can change over time depending upon the incoming request rate from clients.

• **Requests/responses must be indistinguishable.** This requires that the messages be identical in size, given that encryption hides their content.

• **Messages must be transformed at each hop.** If they are not transformed, an observer could track a specific message through the server cloud. Different approaches exist for handling this requirement, with the most extreme being full encryption of each out-

going message. We plan to investigate whether less costly transformation techniques can be employed.

With these requirements in place, a given client request cannot be traced to its destination server node and the subsequent response cannot be traced to its sender. However, this system is still vulnerable to two types of attacks. First, this approach allows information about the performance characteristics of the destination server to leak to passive attackers. For example, if a client has only a single request outstanding, an observer can monitor the time until a response returns, and thus bound the service time of some (unknown) disk. Second, an active attacker can selectively delay or drop packets to determine the destination of a request. For example, the attacker can intercept all packets sent within the cloud except those destined to the intermediary server and a target server; if a response is returned from the intermediary to the client, the attacker can infer that the target server was the destination server.

To guard against these attacks, we regulate the flow of responses to the client from the server cloud. Specifically, the server responds to the client at a pre-established *client response time* (CRT). If the real response has not yet been generated by the CRT (*e.g.*, the disk is quite busy), the responding server sends a dummy response, which in turn generates a dummy request from the client. If the response is still not received at time $2 \times$ CRT, another dummy exchange occurs, and the process continues until the actual response is generated. From the perspective of an attacker, the dummy request and responses cannot be discerned from normal traffic. By monitoring or even interfering with this traffic stream, the attacker is only able to gain minimal information about the storage system, such as a weak bound on the service time of all the disks in the system. Of course, overall storage system performance is sensitive to CRT, as we will explore in the next section. Since the likelihood of these attacks may be small, we consider algorithms with and without this final requirement.

2.5 Adaptative Controllers

Given our basic anonymity architecture, there are two important points of flexibility in its implementation: server-cloud topology and forwarding interval. As we shall discuss, both require *adaptive controllers* to choose the optimal settings given the current workload and hardware configuration.

Topology: The first point of flexibility is the topology of the server cloud. We examine three different topologies: ring, all-to-all, and hypercube. In each topology, the client sends a request to a randomly-selected intermediary server which acts as an entry point to the server cloud. The request is then forwarded along through a series of servers according to the selected topology until it reaches the destination server, at which point, the response takes the appropriate path back to the client.

In the ring topology, at each forwarding interval, each server forwards a request (or a dummy request) to a single neighbor. Each request travels through the ring network until it reaches the destination server. After the destination server handles the request, the reply is propagated on at the next forwarding interval, and so on until it again reaches the intermediary server, which sends the result back to the client (after a potential CRT delay). The problem with the ring topology is poor performance; in a D-server system, each message is forwarded D times.

To address this problem, we consider an all-to-all topology, which represents the other extreme in the topology space. At each forwarding interval, every server in the cloud sends a message (whether real or a dummy) to all $D-1$ other servers. Thus, a request reaches its destination in only two hops. Compared to ring, latency is greatly reduced. However, the high number of neighbors may force extra dummy messages to be sent, wasting network bandwidth.

To provide a middle ground between these extreme topologies, we investigate a hypercube topology. In a hypercube, each server is connected to exactly $lg_2(D)$ neighbors and a message from one server can reach any other server in at most $lg_2(D) - 1$ steps. Thus, at each forwarding interval, each server forwards a real or dummy message to each of its direct $\lg_2(D)$ neighbors. Given that a server has fewer neighbors than in all-to-all, these servers do not need to send as many dummy messages, potentially improving performance. However, because messages need to be forwarded more times, available network bandwidth may be stressed.

The optimal topology for a given system configuration is a function of the client request traffic. The all-to-all topology wastes bandwidth in the form of dummy messages sent when the disk does not have sufficient useful messages to fill its transmit window, whereas in hypercube, the wasted bandwidth stems from the multiple hops each message has to go through. Thus, the all-to-all topology performs better

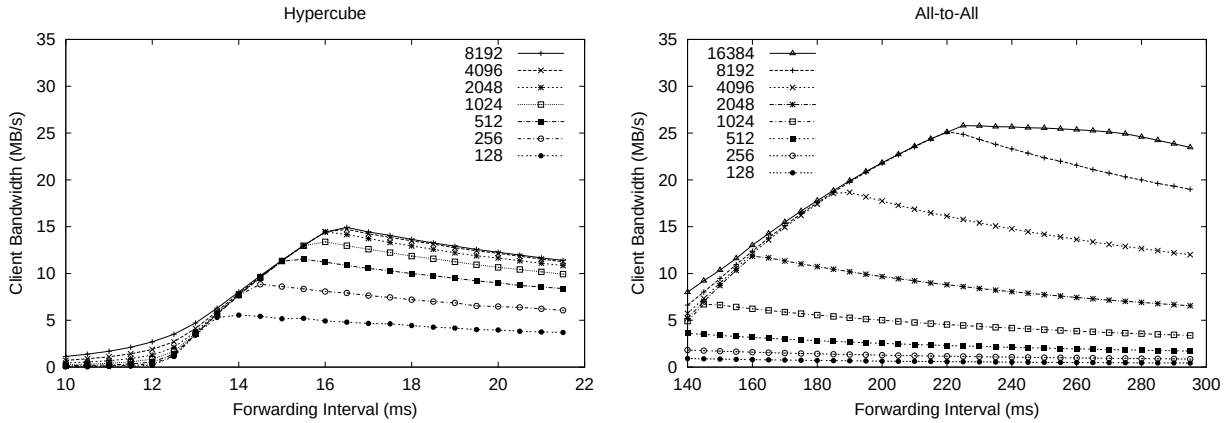

Figure 1: **Sensitivity of Topologies to Forwarding Interval.** *These two graphs illustrate that the optimal forwarding interval is dependent upon the number of messages outstanding from the client. In each graph, the forwarding interval is varied along the x-axis, and the resulting client bandwidth is shown along the y-axis. Each line designates a different number of outstanding messages from the client. The ring topology is not shown due to lack of space.*

when there are sufficient client messages outstanding in the system; if not, the hypercube is better (the ring topology is strictly inferior to both).

To choose the proper topology, an adaptive controller is required, which needs to dynamically select the topology that minimizes the wasted bandwidth given the current client workload. To obtain this information, each server keeps track of the average useful window size (the number of useful messages transmitted during every forwarding interval). Analytically, we find that the hypercube topology performs better than the all-to-all topology when the average window is less than $\frac{2 \cdot D}{\lg_2(D)}$, where D is the number of disks (the proof is given in the appendix). The adaptive topology controller examines the window size periodically and chooses the optimal topology for the current window size.

Forwarding Interval: We now consider the second point of flexibility in our design, the forwarding interval among servers. The forwarding interval determines how much time a server in the cloud waits before forwarding messages to its "neighboring" servers, as determined by the topology in use.

Setting the forwarding interval offers an obvious trade-off. If the forwarding interval is very large, latency increases, since each request takes longer to move through its fixed number of hops; however, in this interval, each server accumulates many real requests and responses, and thus bandwidth utilization is increased (*i.e.,* fewer dummy messages must be sent). On the other hand, if the forwarding interval is very small, useful throughput decreases.

The forwarding-interval controller is driven by client feedback. It operates on the fact that any change to the interval in the right direction is likely to increase client bandwidth, and vice-versa. The controller periodically changes the interval in the current direction (the initial direction is arbitrary) by a certain percentage, and then observes the impact on the client bandwidth. If the impact is positive, it persists in that direction; if the impact is negative for two consecutive samples, it switches direction.

3 Simulation Results

In this section, we use simulations to demonstrate the effectiveness of our adaptive algorithms for providing cost-effective anonymity. We begin by briefly describing our simulation framework and basic assumptions. Next, we motivate the need for algorithms that can adapt the forwarding interval and topology of communication, and then demonstrate that our adaptive algorithms perform as desired. We then explore the sensitivity of our algorithm to the number of clients, the workload, message size, and the client response time threshold.

3.1 Simulation methodology

To explore adaptive anonymity algorithms, we have implemented a simple event-based simulator of a distributed file system. The simulation consists of set of clients and a set of disks, connected with a network containing a single switch. We simulate a "large"

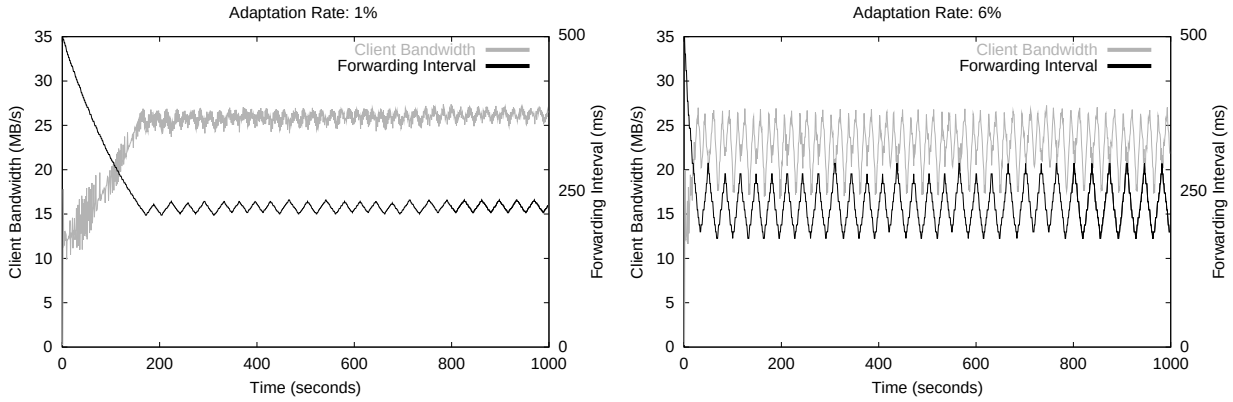

Figure 2: **Sensitivity to Rate of Adaptation for All-to-all Topology.** *Each graph plots two different metrics for a single experiment, the delivered client bandwidth and the internal forwarding delay, over time. The first graph shows that when the adaptation rate is relatively small (i.e., 1%), the system slowly decreases its forwarding delay from its initial value to the best value, but once set, it remains relatively stable along with the resulting bandwidth. In the second graph, where the adaptation is relatively large (i.e., 6%), the system more rapidly tunes its forwarding delay to the best value, but experiences more variation over time in its forward delay and the resulting client bandwidth.*

system, comprised of 64 disks, and accessed by either one or multiple clients. The bandwidth of the switched network in our simulation experiments is set at 1 Gb/s and the disks operate at 20 MB/s.

3.2 Static Anonymity

In our first set of experiments, we examine static anonymity algorithms that forward messages at a fixed, specified rate in the hypercube and all-to-all topologies (the ring topology is strictly worse than both and is not shown due to space considerations). These results as a function of the number of outstanding client requests are shown in Figure 1. From these graphs, we make two observations.

First, for each topology, the bandwidth delivered by the storage system to the client increases as the number of outstanding messages is increased. Increasing the number of outstanding messages has the effect of increasing the rate of requests sent by the client; less network bandwidth is wasted, and thus more disks effectively utilized. In the hypercube topology, the client achieves a peak bandwidth proportional to $\frac{1}{\log_2(D)}$, where D is the number of disks. In the all-to-all topology, the client is able to achieve a peak bandwidth of approximately $\frac{1}{4}$ of the total switch bandwidth, due to the increase in traffic by a factor of four (later we will include an optimization that increases useful traffic to $\frac{1}{2}$). The ring topology (not shown) offers the worst performance, as all messages move through a D-disk system D times.

Second, the optimal topology also varies with the number of outstanding messages. With this system configuration (specifically, number of disks and network bandwidth), the hypercube topology is superior with less than 3000 outstanding client requests while all-to-all is superior with more than 3000. Intuitively, for a given client request rate, a disk in the hypercube topology does not need to wait as long as in the all-to-all topology to have real requests to send all of its neighbors; thus, each disk is able to forward data through the system more rapidly. However, in the hypercube topology, each message must pass through more disks to reach its final destination; thus, when the system has sufficient number of outstanding messages, the all-to-all topology is superior.

In summary, the optimal forwarding interval and the best forwarding topology are both a function of the number of outstanding client requests. Thus the system must adapt both its forwarding interval and forwarding topology to the incoming rate.

3.3 Adaptive Anonymity

We begin by exploring the sensitivity of our adaptive algorithm to the rate at which it changes the forwarding interval. Intuitively, there exists a trade-off between reactivity and stability: the more rapidly the forwarding interval is changed at each disk, the more quickly the system adjusts to changes in the client sending rate, but the less stable is the resulting bandwidth. This trade-off is illustrated in Figure 2.

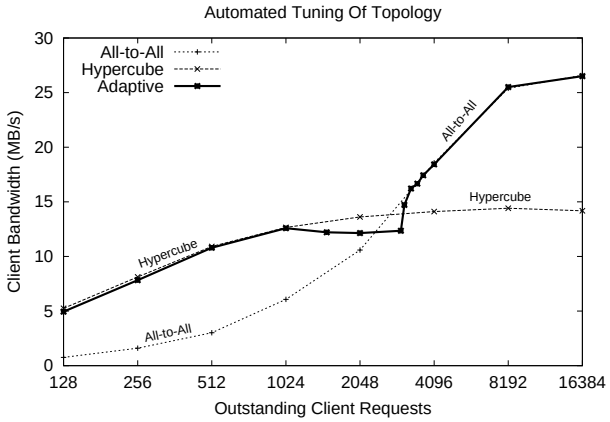

Figure 3: **Performance of Adaptive Algorithms.** *This graph shows the performance of three adaptive algorithms as the number of outstanding client requests is increased. The "All-to-all" and the "Hypercube" lines show client bandwidth when the topology is fixed as all-to-all and hypercube respectively; in both, the forwarding interval is adaptive. The "Adaptive Topology" line shows performance when both topology and forwarding interval are adaptive. Each experiment runs for 1000 seconds.*

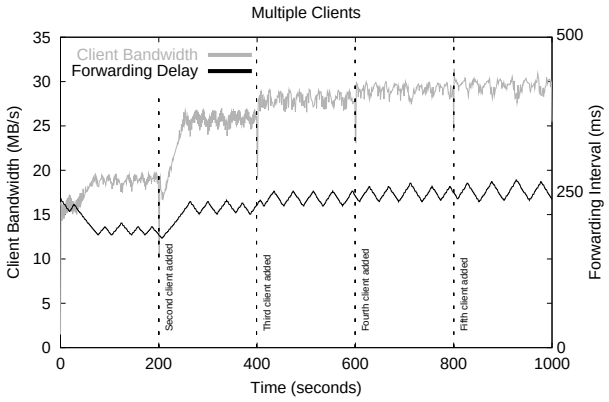

Figure 4: **Sensitivity to Multiple Clients.** *This figure plots the performance of our adaptive algorithm under multiple clients. Every 200 seconds, a new client is added and immediately begins making requests of the servers. The 64 servers respond by adjusting the forwarding interval appropriately. The graph plots the aggregate client bandwidth summed across all clients.*

In this experiment, we examine a single workload with 8192 outstanding messages and set the initial forwarding interval at each disk to 500 ms. In the first graph, the adaptation rate is set relatively low, to 1%. A 1% adaptation rate means that at every adjustment period (roughly every 2 seconds in our simulations), the largest change in forwarding interval that is made is 1% of the current value. As we can see from the figure, with a 1% adaptation rate, the disks take a significant amount of time (nearly 200 seconds) to move their forwarding interval from the initial value of roughly 500 ms (as shown on the right y-axis) to the final near-optimal value of 220 ms. However, once the disks reach this forwarding interval, their chosen interval remains relatively stable as does the resulting client bandwidth. The second graph shows that when the adaptation rate is set to a more aggressive 6%, the disks reach the optimal forwarding interval much more quickly (in roughly 10 seconds), but experience more variation in both their forwarding interval and the client bandwidth. As we believe that workload changes will be slow, we choose the 1% adaptation rate for the rest of the experiments.

The performance of our adaptive algorithm is summarized in Figure 3, in which the delivered client bandwidth is plotted as a function of the number of outstanding client requests. The figure illustrates that when the topology is fixed in a hypercube, for each workload the delivered bandwidth matches the best bandwidth achieved in the first graph of Figure 1; similarly, when the topology is fixed as all-to-all, the delivered bandwidth matches the best delivered bandwidth of the second graph of Figure 1. Finally, when our algorithm adapts its topology as a function of the incoming client rate, it delivers bandwidth similar to the best of each topology.

3.4 Sensitivity to Multiple Clients

Although the experiments presented thus far have assumed a single client accessing a large group of disks, many realistic environments will consist of multiple clients. Thus, we show that our adaptive algorithms also function well with multiple clients.

Figure 4 presents the results of an experiment where clients are introduced to the system over time. At time 0, there is only one client present in the system; at every subsequent 200 second interval, another client is added. All clients generate 4096 outstanding requests. As one can see from the graph, the system has no trouble adapting to the rate of incoming requests, and adjusts its forwarding interval appropriately in each case. The only reason that the aggregate bandwidth of the clients peaks at roughly 30 MB/s is that it is network-limited at that point.

3.5 Sensitivity to Workload

In the experiments presented in this paper up until this point, clients have generated workloads that access files distributed uniformly across all disks. To make sure that our approach does not rely upon client workload in any way, we have examined experiments with skewed workloads, where most data accesses are directed to a subset of the disks in the system.

In experiments not shown here, we found that the behavior of the system with a skewed workload is similar to the uniform workload; most importantly, the servers always exchange the same number of messages, regardless of incoming workload.

3.6 Variable-Size Messages Optimization

The previous experiments have assumed that all messages are padded to fill the maximum message size of 4 KB. However, given the bimodal distribution of file request sizes (read requests are short; read responses are long), this approach wastes half of the network bandwidth. Thus, we explore two different approaches for both the hypercube and all-to-all topologies; the bandwidth delivered to the client is summarized in in Table 1. In the first optimization, we place all communication into 4140 byte messages (*i.e.*, read requests and read responses can share the same message); the second column of the table shows that this significantly improves the delivered bandwidth to the client for both topologies. In the second optimization, we allow the client to communicate with messages that correspond to their true size; for example, a read request message is placed in a small message. While this leaks the type of operation that the client is performing (*i.e.*, a read or a write), it does not allow attackers to discover anything about the internals of the server cloud. With this optimization (third column), client bandwidth is effectively doubled versus the original approach, and for the all-to-all topology, gets to roughly within a factor of two of a system providing no anonymity.

3.7 Sensitivity to Client Response Time

In our final set of experiments, we examine the performance impact of adding safeguards against active attacks; specifically, we measure the delivered client bandwidth and latency as a function of the preestablished client response time (CRT). These results are shown in Figure 5 for the hypercube topology. As indicated in the graph, if the client response time is too small, operations do not have sufficient time to be serviced within that interval, resulting in dummy responses from the intermediary node and consequent dummy requests from the client, thereby wasting network bandwidth. Alternatively, if the client response time is too large, responses are delayed at the intermediary node, directly limiting deliverable bandwidth and increasing operation latency. Thus, the optimal fixed client response time threshold is directly related to the expected service time of each request. We note that the system cannot adapt the client response time threshold as a function of the amount of dropped or delayed messages without leaking information concerning whether real or dummy messages are being interfered with; a fixed threshold must be chosen. Also note that the average response time per request is high, as forwarding delays hurt response time. For applications that cannot hide latency via prefetching and write-behind techniques, anonymity in the presence of active attackers may be too costly.

	Pad	Merge	True	None
Hypercube	15	23	29	125
All-to-all	26	35	52	125

Table 1: **Sensitivity of Bandwidth to Variable-Sized Messages** *The table displays the client bandwidth with both the hypercube and all-to-all topologies as a function of the message size. The workload assumes a single client, and 16K outstanding client requests. The first column assumes all messages are padded to fit in 4 KB, matching the approach used for most experiments in this paper. The second column merges short and long messages into a 4140-byte message, so that a read request and read response can fit in one message. The third column shows the benefit when messages to and from the client correspond to their true size: e.g., read requests are short. The final column shows client bandwidth with no anonymity (the topology is not applicable in this column).*

4 Implementation

We describe and evaluate a prototype anonymous storage implementation. The goal of our preliminary implementation is to demonstrate the feasibility of our basic design, and that its performance is roughly as expected given our previous simulation studies. However, we have made a number of simplifications in our design to complete the initial prototype. In this section, we first describe the development environment, then discuss a few implementation details, and finally present a brief evaluation of the prototype.

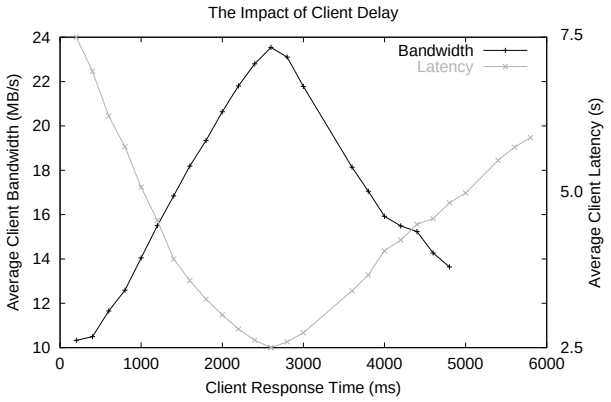

Figure 5: **Sensitivity to Client Response Time.** *This figure plots the average bandwidth and latency as perceived by the client as the Client Response Time (CRT) is varied in order to prevent active attacks from gathering information. This experiment considers a single client connected to a 64-node hypercube topology storage system with 16K outstanding requests. Along the x-axis, the client response time is varied, corresponding to the amount of time each response is held.*

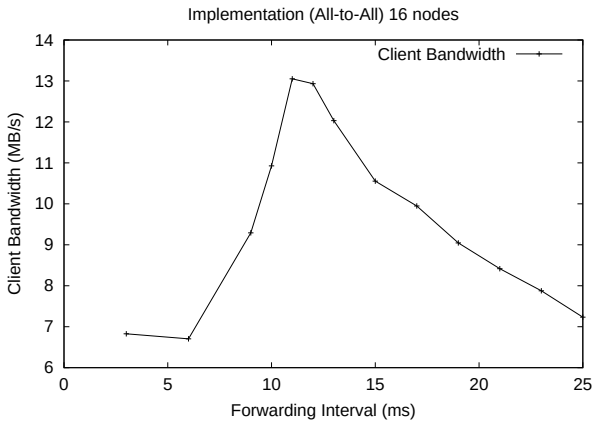

Figure 6: **Implementation: Varying Forwarding Interval.** *The figure shows the effect of varying the forwarding interval on the overall performance of the prototype implementation. For each experiment, the forwarding interval is fixed, and a single client reads 100 MB sequentially from a 16-disk storage system. The y-axis plots throughput achieved during the experiment.*

Our system currently runs on a testbed of 16 Intel-based machines, running the Linux operating system; the machines are connected via Gigabit Ethernet. Each PC contains a 9.1GB IBM Ultrastar 9LZX disk. Communication is performed over RPC on top of UDP/IP. The prototype is implemented as a loadable file system in the Linux 2.2 kernel. Protocols required to establish and maintain secure communication channels between clients, disks, and the file manager are implemented as described in Section 2.

We make a number of simplifications in our implementation. Most importantly, our system does not yet perform adaptation of either topology or forwarding interval; currently, we assume an all-to-all topology and manually vary the forwarding interval. We also do not yet implement the client response time (CRT) dummy exchange. However, we believe the prototype is still useful in verifying that our anonymity protocols behave in a real implementation as they do in simulation. We are working on a more complete prototype.

Figure 6 presents the performance of the prototype. The graph plots the performance of a 100 MB file read, while varying the forwarding interval along the x-axis (the forwarding interval is fixed for each particular experiment). As we can see from the figure, setting the forwarding interval correctly is crucial for performance in the prototype, and the general behavior is as we would expect from simulation.

5 Related Work

There has been a fair amount of security work in storage systems, although most of it is based on ensuring integrity of communication or privacy. An excellent survey by Riedel *et al.* covers much of the work in this area [14]. In early work on NASD systems, Gobioff *et al.* argued that for performance reasons, only message integrity can be guaranteed [9]; however, recent work by Miller *et al.* demonstrates that privacy of each request and reply is possible without too great of a performance degradation [10]. Considering CPU trends, we believe that an increasing amount of processing time can be spent in providing security for storage systems.

There has been a tremendous amount of work on anonymous communication and storage, much of which has informed and inspired our work [2, 3, 4, 5, 6, 11, 12, 13]. However, most of these efforts focus on hiding user identity in a wide-area setting. We instead show the efficacy of anonymous protocols in hiding characteristics of the system itself, and apply anonymous protocols to a local-area network environment. A key difference is that schemes that assume an attacker cannot monitor all communication in the system are not applicable in the local-area.

Early work by Chaum introduced the notion of a *mix*, which acts as an anonymizing intermediary between senders and receivers [3]. In our system, we arrange for the disks themselves to act as a type of mix; however, unlike mixes, increasing the length of forwarding does not improve anonymity.

Pfitzmann and Waidner discuss the concepts of sender anonymity, receiver anonymity, and sender-receiver linking [12]. In our work, we concentrate primarily on receiver anonymity (*i.e.*, hiding the identity of the disk to which a request is sent, as well as the characteristics of the disk itself) and on the prevention of sender-receiver linking.

The levels of anonymity introduced by Reiter and Rubin in their Crowds work is also of interest [13]. They discuss a taxonomy ranging from provably exposed to absolute privacy, and provide proofs of their system's level of anonymity. In the future, we hope to apply similar proofs to our work.

Finally, our hypercube topology bears similarity to the work on anonymous broadcast in Xor-trees [7].

6 Conclusions

We have studied the addition of anonymity into a network-attached storage system. Anonymity is useful in such an environment primarily because it hides the activity of the system from network attackers. By keeping information as to the location of client requests, the identity of the file manager, and the timing characteristics hidden, anonymity prevents focused attacks on the storage system.

Through simulations, we find that anonymity can be added with a relatively modest constant-factor increase in network load. For example, in the all-to-all topology, our anonymity scheme increases traffic by a factor of two as compared to a non-anonymous storage system. However, the per-block latency of anonymous storage is likely to be significantly higher than a more traditional storage architecture; therefore, applications that cannot tolerate such latencies may not be suited for an anonymous environment.

We also find that adaptation helps in moving difficult decisions from the human administrator into the control of the system itself. By adapting the topology and forwarding interval to the workload, peak bandwidth can be achieved.

Finally, we demonstrate that a prototype implementation of an anonymous storage system is likely to behave similarly to the simulations we presented. Thus, our simulations and implementation both point to the feasibility of adding cost-effective, adaptive anonymity to network-attached storage.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *OSDI '02*, Boston,MA, 2002.

[2] O. Berthold, H. Federrath, and M. Köhntopp. Project "Anonymity and Unobservability in the Internet". In *Workshop on Freedom and Privacy by Design / CFP2000*, 2000.

[3] D. L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Workshop on Design Issues in Anonymity and Unobservability*, December 2000.

[5] D. A. Cooper and K. P. Birman. Preserving privacy in a network of mobile computers. In *IEEE Symposium on Security and Privacy*, 1995.

[6] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, December 2000.

[7] S. Dolev and R. Ostrovsky. Xor-trees for effi cient anonymous multicast and reception. Technical Report 98-54, DIMACS, December 1998.

[8] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka. File Server Scaling with Network-Attached Secure Disks. In *SIGMETRICS '97*, pages 272–284, Seattle, WA, June 1997.

[9] H. Gobioff, G. Gibson, and D. Tygar. Security for Network Attached Storage Devices. Technical Report CMU-CS-97-185, Carnegie Mellon, 1997.

[10] E. L. Miller, W. E. Freeman, D. D. Long, and B. C. Reed. Strong Security for Network-Attached Storage. In *FAST '02*, pages 1–13, Monterey, CA, January 2002.

[11] A. Pfi tzmann, B. Pfi tzmann, and M. Waidner. ISDN-Mixes: Untraceable Communication with Very Small Bandwidth Overhead. In *GI/ITG Conference: Communication in Distributed Systems*, Mannheim, February 1991.

[12] A. Pfi tzmann and M. Waidner. Networks Without User Observability – Design Options. In F. Pichler, editor, *Advances in Cryptology: Eurocrypt 85*, volume 219, pages 245–253, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1986. Springer-Verlag.

[13] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[14] E. Riedel, M. Kallahalla, and R. Swaminathan. A Framework for Evaluating Storage System Security. In *FAST '02*, Monterey, CA, January 2002.

[15] J. Schindler and G. R. Ganger. Automated Disk Drive Characterization. Technical Report CMU-CS-99-176, Carnegie Mellon, 1999.

A APPENDIX

A.1 Privacy Infrastructure

Since the content of messages transmitted across the network often reveals substantial information relating to the endpoints as well as the nature of communication taking place, privacy of data becomes a prerequisite for anonymity. The mechanism we use to enforce privacy of communication between the clients and servers is similar to the scheme described by Miller *et al.* [10]. Data is stored encrypted on the disks, thereby relieving the disks of the expense of encrypting and decrypting data (although data may be transformed again in various anonymity schemes, as described below). A per-file access key is used to encrypt data in the file. The access key is maintained at the file manager as part of the meta-data for the file, and is distributed in a secure manner to users who share access to the file. The private key of a client, obtained by a one-way hash on the user's password, serves as the shared secret key for the secure channel between the client and the file manager.

To read a file, a client sends a lookup request to the file manager encrypted in its private key. If the client is authorized to access the file (as indicated by the file's ACL), the file manager sends back the file meta-data including the file access key, encrypted in the private key of the client. A successful lookup also returns capabilities that permit the client to access individual objects on the disks that constitute the file. A detailed description of the capability scheme is described by Gobioff *et al.* [9]. Having both the access key to the file and the required capabilities to access that data at the disks, the client can now access the appropriate data, performing any encryption or decryption at its end. The disks are oblivious to whether the objects it stores are encrypted or not. It is to be noted here that access keys and capabilities are cached in the client kernel; therefore, message exchanges with the file manager are quite infrequent.

A.2 Threshold for adapting topology

To estimate the size of the average useful window size below which the all-to-all topology is inferior to hypercube, we perform an analysis of the total useful bandwidth that is delivered out of the server cloud. For every client request that enters the server cloud, the all-to-all topology incurs two *extra* messages, one to forward the request to the destination and the other to get the reply back from the destination. In other words, for every two *real* messages exchanged within the server cloud, one client request is served. Similarly, in the hypercube topology, to serve one client request, a total of $\lg_2(D)$ messages must be exchanged through the server cloud.

Let T_a and T_h be the optimal forwarding delays for a certain client workload in the all-to-all and hypercube topologies respectively. Let w be the average *useful window size*, i.e. number of real messages sent by a disk at each forwarding interval. Since the number of neighbors for each disk in the hypercube topology is very small, $w = \lg_2(D)$ for a hypercube (i.e. all messages transmitted are real). Since each disk transmits w real messages during every forwarding interval,

Inter-disk message rate in all to all = $\frac{D.w}{T_a}$

Inter-disk message rate in hypercube = $\frac{D.\lg_2(D)}{T_h}$

Since the all-to-all topology requires two inter-disk messages to service one client request, and a hypercube requires $\lg_2(D)$ inter-disk messages,

Useful client bandwidth in all to all = $\frac{D.w}{2.T_a}$

Useful client bandwidth in hypercube = $\frac{D}{T_h}$

Since each disk transmits to $\lg_2(D)$ neighbors in hypercube, versus D neighbors in all-to-all, assuming that the disks do just enough communication to saturate the server network,

$$T_h = \frac{\lg_2(D)}{D}.T_a$$

Useful client bandwidth in hypercube = $\frac{D^2}{\lg_2(D).T_a}$

Thus the all-to-all topology is better than hypercube if

$$\frac{D.w}{2.T_a} > \frac{D^2}{\lg_2(D).T_a}$$

$$w > \frac{2.D}{lg_2(D)}$$